# Opis projektu WCF wykonanego na zaliczenie z przedmiotu Programowanie współbieżne i rozproszone

**Informatyka, studia II stopnia, rok akademicki 2019/2020**

**Studia niestacjonarne**

**Imię i nazwisko studenta: grupa: I\_K-ce\_19\_z\_NII\_gc01\_ADZD**

Kamil Nowak

**Temat projektu:**

Program NorthwindBusinessPartnerIndex udostępniający operacje CRUD dla tabel kontrahentów bazy Northwind.

## Cele projektu

Projekt ma na celu zaprezentowanie użycia technologii WCF w połączeniu z Entity Framework oraz z klientem WPF. Operacje CRUD (Create Read Update Delete) zostały zaimplementowane dla tabel kontrahentów z bazy Northwind – tabeli klientów (Customers), tabeli dostawców (Suppliers) i tabeli przewoźników (Shippers). Całość powinna umożliwiać odczyt, edycję, usuwanie istniejących i dodawanie nowych danych.

## Serwer

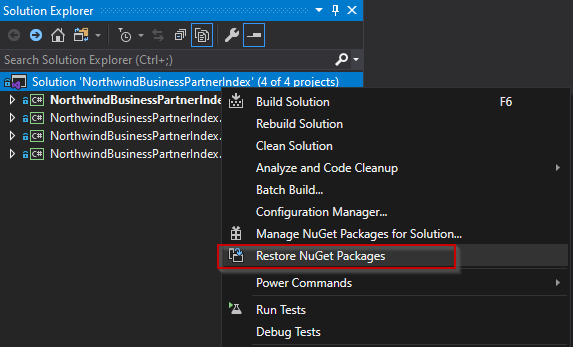
### Wymagania sprzętowe i systemowe oraz biblioteki

Projekt WCF wykorzystuje .NET Framework w wersji 4.7.2. Wykorzystane zostały biblioteki:

* Entity Framework (wersja 6.0) – udostępniająca operacje na bazie danych MS SQL Server

Połączenie z bazą danych było testowane na MS SQL Server 2017 (RTM) - 14.0.1000.169 (X64) – dla nowszych wersji program powinien działać bez problemów jednak nie był testowany i zaleca się zainstalować dokładnie tą samą wersję. Aby odtworzyć bazę danych Northwind należy zainstalować wcześniej wymienioną (lub nowszą) wersję MS SQL Server oraz wykonać plik Northwind.sql (załączony razem z plikami projektu). Jest to anglojęzyczna wersja bazy i tylko z taką będzie działać projekt. Jeżeli program zostanie uruchomiony przed odtworzeniem bazy, baza powinna zostać utworzona automatycznie jednak wtedy będzie ona pusta – zaleca się najpierw odtworzenie bazy a dopiero później uruchomienie programu. Aby program prawidłowo łączył się z bazą należy podać nazwę serwera w pliku konfiguracyjnym App.config w projekcie NorthwindBusinessPartnerIndex.Host w miejscu zaznaczonym na poniższym screenie:



Aby wszystkie zależności zostały zaimportowane wystarczy w Visual Studio kliknąć prawym przyciskiem na solucji i wybrać przywrócenie pakietów NuGet (jak na załączonym poniżej screenie): 

### Kontrakt

Wszystkie kontrakty zostały w obrębie solucji Visual Studio zostały umieszczone w osobnym projekcie o nazwie NorthwindBusinessPartnerIndex.Contracts,

W celu ułatwienia pewnych operacji wydzielono wspólny interfejs implementowany przez kontrakty danych oraz przez klasy encji.

public interface IBusinessPartner

{

string Id { get; }

string CompanyName { get; set; }

}

Wydzielono również generyczny interfejs kontraktu implementowany przez wszystkie kontrakty dla poszczególnych typów kontrahentów:

[ServiceContract]

public interface IDataService<T> where T : IBusinessPartner

{

[OperationContract]

bool AddOrUpdate(T entity);

[OperationContract]

bool Delete(T entity);

}

[ServiceContract]

public interface ICustomerService : IDataService<CustomerDto>

{

[OperationContract]

IList<CustomerDto> GetAllCustomers();

[OperationContract]

CustomerDto GetCustomerById(int id);

}

[ServiceContract]

public interface IShipperService : IDataService<ShipperDto>

{

[OperationContract]

IList<ShipperDto> GetAllShippers();

[OperationContract]

ShipperDto GetShipperById(int id);

}

[ServiceContract]

public interface ISupplierService : IDataService<SupplierDto>

{

[OperationContract]

IList<SupplierDto> GetAllSuppliers();

[OperationContract]

SupplierDto GetSupplierById(int id);

}

Kontrakty danych:

[DataContract]

public class CustomerDto : IBusinessPartner

{

[DataMember]

public string Id { get => CustomerID; set => CustomerID = value; }

[DataMember]

public string CustomerID { get; set; }

[DataMember]

public string CompanyName { get; set; }

[DataMember]

public string ContactName { get; set; }

[DataMember]

public string ContactTitle { get; set; }

[DataMember]

public string Address { get; set; }

[DataMember]

public string City { get; set; }

[DataMember]

public string Region { get; set; }

[DataMember]

public string PostalCode { get; set; }

[DataMember]

public string Country { get; set; }

[DataMember]

public string Phone { get; set; }

[DataMember]

public string Fax { get; set; }

}

[DataContract]

public class ShipperDto : IBusinessPartner

{

[DataMember]

public string Id

{

get => ShipperID.ToString();

set

{

int.TryParse(value, out int result);

ShipperID = result;

}

}

[DataMember]

public int ShipperID { get; set; }

[DataMember]

public string CompanyName { get; set; }

[DataMember]

public string Phone { get; set; }

}

[DataContract]

public class SupplierDto : IBusinessPartner

{

[DataMember]

public string Id

{

get => SupplierID.ToString();

set

{

int.TryParse(value, out int result);

SupplierID = result;

}

}

[DataMember]

public int SupplierID { get; set; }

[DataMember]

public string CompanyName { get; set; }

[DataMember]

public string ContactName { get; set; }

[DataMember]

public string ContactTitle { get; set; }

[DataMember]

public string Address { get; set; }

[DataMember]

public string City { get; set; }

[DataMember]

public string Region { get; set; }

[DataMember]

public string PostalCode { get; set; }

[DataMember]

public string Country { get; set; }

[DataMember]

public string Phone { get; set; }

[DataMember]

public string Fax { get; set; }

[DataMember]

public string HomePage { get; set; }

}

### Implementacja kontraktu

Wszystkie kontrakty zostały w obrębie solucji Visual Studio zostały umieszczone w projekcie NorthwindBusinessPartnerIndex.Host. Kod implementacji poszczególnych kontraktów jest bardzo zbliżony do siebie, wszystkie z nich przyjmują klasę typu UnitOfWork, która udostępnia metody.

W tym projekcie wzorzec UnitOfWork nie został w pełni zaimplementowany (rolą tego wzorca jest zebranie operacji na kilku repozytoriach i wysłanie ich w formie jednej transakcji do bazy danych), tutaj operacje są wysyłane od razu po ich wywołaniu - klasa UnitOfWork jest wykorzystywana tylko jako klasa zbierająca wszystkie implementacje repozytoriów w jednym miejscu. Rolą repozytoriów z kolei jest odseparowanie warstwy dostępu do danych EntityFramework od warstwy aplikacji. Wszelkie klasy i interfejsy powiązane z warstwą dostępu do danych zostały umieszczone w projekcie NorthwindBusinessPartnerIndex.Data.

Implementacje kontraktów korzystają również z klasy Mapper, która udostępnia metody konwertujące obiekty encji na obiekty transferu danych DTO (Data Transfer Object – czyli tutaj klasy kontraktów danych) wykorzystywane przy operacjach odczytu, oraz metody konwertujące obiekty dto na encje, wykorzystywane przy operacjach zapisu.

public class CustomerService : ICustomerService

{

private readonly UnitOfWork \_unitOfWork;

public CustomerService(UnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public IList<CustomerDto> GetAllCustomers()

{

var customers = \_unitOfWork.Customers.GetAll().ToList();

return customers.Select(entity => Mapper.Map(entity)).ToList();

}

public CustomerDto GetCustomerById(int id)

{

var entity = \_unitOfWork.Customers.Get(id.ToString());

return Mapper.Map(entity);

}

public bool AddOrUpdate(CustomerDto dto)

{

var result = \_unitOfWork.AddOrUpdate(Mapper.Map(dto));

if (result)

{

\_unitOfWork.Commit();

}

return result;

}

public bool Delete(CustomerDto dto)

{

var result = \_unitOfWork.Delete(Mapper.Map(dto));

if (result)

{

\_unitOfWork.Commit();

}

return result;

}

}

Wszystkie kontrakty zostały zagregowane w klasie BusinessPartnerService (fragment klasy poniżej).

Dodatkowym zadaniem BusinessPartnerService jest zapis informacji o wykonywanych operacjach w oknie konsoli.

public class BusinessPartnerService : ICustomerService, ISupplierService, IShipperService

{

private readonly CustomerService \_customerService;

private readonly ShipperService \_shipperService;

private readonly SupplierService \_supplierService;

public BusinessPartnerService(

CustomerService customerService,

ShipperService shipperService,

SupplierService supplierService)

{

\_customerService = customerService;

\_shipperService = shipperService;

\_supplierService = supplierService;

}

public IList<CustomerDto> GetAllCustomers()

        {

           Console.WriteLine("Get all customers...");

           var result = \_customerService.GetAllCustomers();

           Console.WriteLine($"\tFetched {result.Count} customers");

           return result;

        }

…

Ponieważ wszystkie serwisy są zależne od UnitOfWork, musiała zostać stworzona niestandardowa implementacja klasy ServiceHost, która pozwala zdefiniować przekazać instancję UnitOfWork do BusinessPartnerServiceInstanceProvider odpowidającej za tworzenie instancji kontraktów.

public class BusinessPartnerServiceHost : ServiceHost

{

public BusinessPartnerServiceHost(UnitOfWork unitOfWork, Type type, params Uri[] address)

: base(type, address)

{

foreach (var cd in ImplementedContracts.Values)

{

cd.Behaviors.Add(new BusinessPartnerServiceInstanceProvider(unitOfWork));

}

}

}

Fragment klasy BusinessPartnerServiceInstanceProvider:

public class BusinessPartnerServiceInstanceProvider : IInstanceProvider, IContractBehavior

{

private readonly UnitOfWork \_unitOfWork;

public BusinessPartnerServiceInstanceProvider(UnitOfWork unitOfWork)

{

\_unitOfWork = unitOfWork;

}

public object GetInstance(InstanceContext instanceContext, Message message)

{

return GetInstance(instanceContext);

}

public object GetInstance(InstanceContext instanceContext)

{

return new BusinessPartnerService(new CustomerService(\_unitOfWork),

new ShipperService(\_unitOfWork), new SupplierService(\_unitOfWork));

}

…

}

### Endpoint

<services>

<service name="NorthwindBusinessPartnerIndex.Host.BusinessPartnerService"

behaviorConfiguration="mexBehaviour">

<endpoint

address="Customers" binding="basicHttpBinding"

contract="NorthwindBusinessPartnerIndex.Contracts.API.ICustomerService"/>

<endpoint

address="Shippers" binding="basicHttpBinding"

contract="NorthwindBusinessPartnerIndex.Contracts.API.IShipperService"/>

<endpoint

address="Suppliers" binding="basicHttpBinding"

contract="NorthwindBusinessPartnerIndex.Contracts.API.ISupplierService"

/>

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

<host>

<baseAddresses>

<add baseAddress="http://localhost:8080/"/>

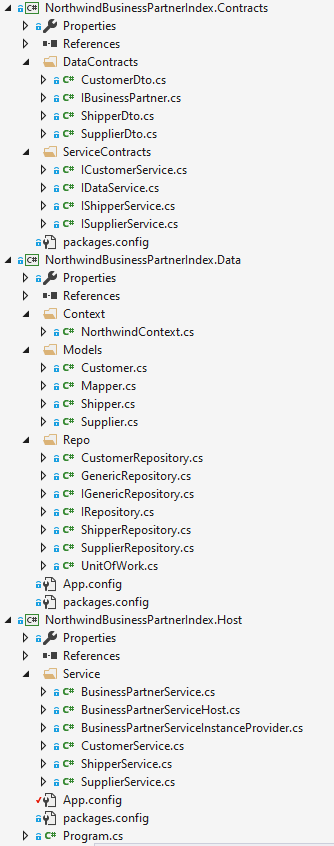
</baseAddresses>

</host>

</service>

</services>

### Pliki projektu (serwer)



## Klient

### Wymagania sprzętowe i systemowe oraz biblioteki

Projekt WPF wykorzystuje .NET Framework w wersji 4.7.2. Wykorzystane zostały biblioteki

* Caliburn.Micro (wersja 3.2) – wykorzystana do implementacji wzorca MVVM (Model-View-ViewModel)

Dokumentacja: <https://caliburnmicro.com/documentation/>

Kod źródłowy: <https://github.com/Caliburn-Micro/Caliburn.Micro>

* Autofac (wersja 5.2) – wykorzystana do implementacji wzorca IoC (Inversion of Control)

Dokumentacja: <https://autofaccn.readthedocs.io/en/latest/>

Kod źródłowy: <https://github.com/autofac/Autofac>

Do konfiguracji aplikacji wykorzystywana jest klasa dziedzicząca po klasie BootstrapperBase (klasa z biblioteki Caliburn.Micro). Jest ona punktem startowym programu i referencja do konkretnej implementacji (tutaj klasa AppBootstrapper) musi zostać podana w pliku App.xml (poniżej fragment pliku).

<Application x:Class="NorthwindBusinessPartnerIndex.Client.App"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:local="clr-namespace:NorthwindBusinessPartnerIndex.Client">

    <Application.Resources>

        <ResourceDictionary>

            <ResourceDictionary.MergedDictionaries>

                <ResourceDictionary>

                    <local:AppBootstrapper x:Key="bootstrapper" />

                </ResourceDictionary>

            </ResourceDictionary.MergedDictionaries>

Program po uruchomieniu tworzy instancje klasy AppBootstrapper. (poniżej fragment klasy)

public class AppBootstrapper : BootstrapperBase

{

public IContainer Container { get; private set; }

public AppBootstrapper()

{

ViewLocator.AddSubNamespaceMapping(

"NorthwindBusinessPartnerIndex.Client.UI.ViewModels",

"NorthwindBusinessPartnerIndex.Client.UI.Views");

Container = BuildContainer();

Initialize();

}

private IContainer BuildContainer()

{

var builder = new ContainerBuilder();

builder.RegisterType<ShellViewModel>().AsSelf().SingleInstance();

builder.RegisterType<MainViewModel>().AsSelf().SingleInstance();

builder.RegisterType<BusinessPartnerListViewModel>().AsSelf().SingleInstance();

builder.RegisterType<BusinessPartnerDataViewModel>().AsSelf().SingleInstance();

builder.RegisterType<CustomerService>().AsSelf().SingleInstance();

builder.RegisterType<ShipperService>().AsSelf().SingleInstance();

builder.RegisterType<SupplierService>().AsSelf().SingleInstance();

builder.RegisterType<BusinessPartnerService>().AsSelf().SingleInstance();

builder.Register<IWindowManager>(c => new WindowManager()).InstancePerLifetimeScope();

builder.Register<IEventAggregator>(c => new EventAggregator())

.InstancePerLifetimeScope();

return builder.Build();

}

W konstruktorze klasy deklarowana jest przestrzeń nazw, w której znajdują się wszystkie ViewModele, (czyli klasy przechowujące logikę biznesową widoku), oraz przestrzeń nazw widoków (czyli plików .xaml). Pozwoli to w trakcie działania aplikacji tworzyć powiązania klas ViewModeli z widokami, na podstawie konwencji nazewnictwa (każda klasa ViewModelu musi mieć w nazwie na końcu ‘ViewModel’, klasa widoku powinna nazywać się tak samo, przy czym w nazwie na końcu powinna mieć ‘View’). Z wykorzystaniem mechanizmów Caliburn.Micro w plikach .xaml dzięki utworzonym powiązaniom można tworzyć widoki na podstawie powiązania z instancję ViewModelu.

Dalej w konstruktorze są rejestrowane zależności za pomocą kontenera IoC z biblioteki Autofac. Przeciążone metody z klasy BoostrapperBase umożliwiają wskazanie wcześniej zbudowanego kontenera jako źródła zależności, oraz w metodzie OnStartup wskazanie widoku, który będzie wyświetlony jako pierwszy. Po tak skonfigurowanej aplikacji wystarczy, że w konstruktorze dowolnej klasy wskażemy jakiego typu zależności powinny zostać wstrzyknięte, a kontener IoC sam zainicjalizuje wszystkie klasy wedle konfiguracji i przekaże je do konstruktora naszej klasy.

Reszta metod klasy AppBootstrapper:

protected override object GetInstance(Type serviceType, string key)

{

if (string.IsNullOrWhiteSpace(key))

{

if (Container.IsRegistered(serviceType))

return Container.Resolve(serviceType);

}

else

{

if (Container.IsRegistered(serviceType))

return Container.Resolve(serviceType);

}

throw new Exception(

string.Format("Could not locate any instances of contract {0}.", key ?? serviceType.Name));

}

protected override IEnumerable<object> GetAllInstances(Type serviceType)

{

return Container.Resolve(typeof(IEnumerable<>)

.MakeGenericType(serviceType)) as IEnumerable<object>;

}

protected override void BuildUp(object instance)

{

Container.InjectProperties(instance);

}

protected override void OnStartup(object sender, StartupEventArgs args)

{

DisplayRootViewFor<ShellViewModel>(new Dictionary<string, object>()

{

{ nameof(Window.Title), "Northwind Business Partner Index" }

});

}

protected override void OnExit(object sender, EventArgs e)

{

base.OnExit(sender, e);

}

### Połączenie z serwerem i endpoint

Podobnie jak w projekcie serwera, klient wykorzystuje osobne klasy do obsługi połączenia dla każdego typu kontrahenta. Każda z klas serwisów dziedziczy po klasie BaseService. Również tutaj powstała klasa agregująca wszystkie serwisy.

public abstract class BaseService<TService, TData>

 where TService : IDataService<TData> where TData : IBusinessPartner

{

protected abstract string Address { get; }

public async Task<bool> AddOrUpdate(TData entity)

=> await FromService(service => service.AddOrUpdate(entity));

public async Task<bool> Delete(TData entity)

=> await FromService(service => service.Delete(entity));

public abstract Task<IList<TData>> GetAll();

public abstract Task<TData> GetById(int id);

public async Task<TResult> FromService<TResult>(Func<TService, TResult> func)

{

var retval = default(TResult);

await Task.Run(() =>

{

var client = new ChannelFactory<TService>(

new BasicHttpBinding(), new EndpointAddress(Address));

try

{

var channel = client.CreateChannel();

retval = func(channel);

client.Close();

}

catch

{

client.Abort();

}

});

return retval;

}

}

public class CustomerService : BaseService<ICustomerService, CustomerDto>

{

protected override string Address => "http://localhost:8080//Customers";

public override Task<IList<CustomerDto>> GetAll()

=> FromService(service => service.GetAllCustomers());

public override Task<CustomerDto> GetById(int id)

=> FromService(service => service.GetCustomerById(id));

}

public class ShipperService : BaseService<IShipperService, ShipperDto>

{

       protected override string Address => "http://localhost:8080//Shippers";

       public override Task<IList<ShipperDto>> GetAll() =>

FromService(service => service.GetAllShippers());

       public override Task<ShipperDto> GetById(int id) =>

FromService(service => service.GetShipperById(id));

}

public class SupplierService : BaseService<ISupplierService, SupplierDto>

{

        protected override string Address => "http://localhost:8080//Suppliers";

        public override Task<IList<SupplierDto>> GetAll() =>

FromService(service => service.GetAllSuppliers());

        public override Task<SupplierDto> GetById(int id) =>

FromService(service => service.GetSupplierById(id));

}

public class BusinessPartnerService

    {

        public CustomerService CustomersService { get; }

        public ShipperService ShippersService { get; }

        public SupplierService SuppliersService { get; }

        public BusinessPartnerService(

CustomerService customersService,

ShipperService shippersService,

SupplierService suppliersService)

        {

            CustomersService = customersService;

            ShippersService = shippersService;

            SuppliersService = suppliersService;

        }

        public async Task<bool> AddOrUpdate<T>(T entity)

        {

            switch (entity)

            {

                case CustomerDto x: return await CustomersService.AddOrUpdate(x);

                case SupplierDto x: return await SuppliersService.AddOrUpdate(x);

                case ShipperDto x: return await ShippersService.AddOrUpdate(x);

                default: return false;

            }

        }

        public async Task<bool> Delete<T>(T entity)

        {

            switch (entity)

            {

                case CustomerDto x: return await CustomersService.Delete(x);

                case SupplierDto x: return await SuppliersService.Delete(x);

                case ShipperDto x: return await ShippersService.Delete(x);

                default: return false;

            }

        }

    }

### Interfejs WPF – kod XAML

Widok okna aplikacji - ShellView:

<Window x:Class="NorthwindBusinessPartnerIndex.Client.UI.Views.ShellView"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:cal="http://www.caliburnproject.org"

        mc:Ignorable="d"

        Title="ShellView"

        WindowStartupLocation="CenterScreen"

        Height="600"

        Width="900">

    <Grid Margin="20">

        <Grid.RowDefinitions>

            <RowDefinition Height="\*" />

            <RowDefinition Height="12\*" />

            <RowDefinition Height="\*" />

        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="\*" />

            <ColumnDefinition Width="12\*" />

            <ColumnDefinition Width="\*" />

        </Grid.ColumnDefinitions>

        <ContentControl

            Grid.Row="1"

            Grid.Column="1"

            cal:View.Model="{Binding MainView}" />

    </Grid>

</Window>

Caliburn.Micro udostępnia właściwość View.Model dla elementu ContentControl, którą można powiązać z instancją klasy ViewModelu odpowiadającej widokowi, który chcemy wstawić.

public class ShellViewModel : Screen

{

public MainViewModel MainView { get; }

public ShellViewModel(MainViewModel mainView)

{

MainView = mainView;

}

}

Widok główny - MainView, który zawiera w sobie widok listy, oraz widok szczegółów

<UserControl x:Class="NorthwindBusinessPartnerIndex.Client.UI.Views.MainView"

              xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:cal="http://www.caliburnproject.org"

             mc:Ignorable="d"

             d:DesignHeight="500"

             d:DesignWidth="900">

    <Grid>

        <Grid.RowDefinitions>

            <RowDefinition Height="\*" />

            <RowDefinition Height="9\*" />

        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="\*" />

            <ColumnDefinition Width="\*" />

            <ColumnDefinition Width="\*" />

            <ColumnDefinition Width="0.5\*" />

            <ColumnDefinition Width="3\*" />

        </Grid.ColumnDefinitions>

        <RadioButton

            Style="{StaticResource {x:Type ToggleButton}}"

            GroupName="BusinessPartners"

            Grid.Column="0"

            Content="Customers"

            Typography.Capitals="AllSmallCaps"

            cal:Message.Attach="[Click]=[ShowCustomers]"

            Margin="3" />

        <RadioButton

            Style="{StaticResource {x:Type ToggleButton}}"

            GroupName="BusinessPartners"

            Grid.Column="1"

            Content="Shippers"

            Typography.Capitals="AllSmallCaps"

            cal:Message.Attach="[Click]=[ShowShippers]"

            Margin="3" />

        <RadioButton

            Style="{StaticResource {x:Type ToggleButton}}"

            GroupName="BusinessPartners"

            Grid.Column="2"

            Typography.Capitals="AllSmallCaps"

            cal:Message.Attach="[Click]=[ShowSuppliers]"

            Content="Suppliers"

            Margin="3" />

        <ContentControl

            Grid.Row="1"

            Grid.Column="0"

            Grid.ColumnSpan="3"

            cal:View.Model="{Binding BusinessPartnerList}" />

        <ContentControl

            Grid.Row="0"

            Grid.RowSpan="2"

            Grid.Column="4"

            cal:View.Model="{Binding BusinessPartnerData}" />

    </Grid>

</UserControl>

Widok listy:

<UserControl x:Class="NorthwindBusinessPartnerIndex.Client.UI.Views.BusinessPartnerListView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             mc:Ignorable="d"

             d:DesignHeight="450"

             d:DesignWidth="800">

    <Border BorderBrush="Black"

            BorderThickness="1">

        <ListBox ItemsSource="{Binding Data, Mode=OneWay}"

                 SelectedItem="{Binding SelectedBusinessPartner}"

                 VerticalAlignment="Stretch"

                 HorizontalAlignment="Stretch">

            <ListBox.ItemsPanel>

                <ItemsPanelTemplate>

                    <StackPanel Orientation="Vertical"

                                Margin="20,0" />

                </ItemsPanelTemplate>

            </ListBox.ItemsPanel>

            <ListBox.ItemTemplate>

                <DataTemplate>

                    <StackPanel>

                        <TextBlock Text="{Binding Path= CompanyName}"

                                   FontSize="12"

                                   Foreground="Black"

                                   FontWeight="Bold" />

                        <Separator BorderThickness="2"

                                   BorderBrush="Black" />

                    </StackPanel>

                </DataTemplate>

            </ListBox.ItemTemplate>

        </ListBox>

    </Border>

</UserControl>

Widok szczegółów:

<UserControl x:Class="NorthwindBusinessPartnerIndex.Client.UI.Views.BusinessPartnerDataView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:NorthwindBusinessPartnerIndex.Client.UI"

             xmlns:cal="http://www.caliburnproject.org"

             mc:Ignorable="d"

             d:DesignHeight="450"

             d:DesignWidth="800">

    <UserControl.Resources>

        <local:BusinessPartnerDataTemplateSelector x:Key="TemplateSelector" />

    </UserControl.Resources>

    <Border

        BorderBrush="Black"

        BorderThickness="1">

        <Grid>

            <Grid.RowDefinitions>

                <RowDefinition Height="14\*" />

                <RowDefinition Height="\*" />

            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>

                <ColumnDefinition Width="\*" />

                <ColumnDefinition Width="\*" />

                <ColumnDefinition Width="\*" />

            </Grid.ColumnDefinitions>

            <ContentControl

                Grid.Row="0"

                Grid.ColumnSpan="3"

                Content="{Binding SelectedBusinessPartner}"

                ContentTemplateSelector="{StaticResource TemplateSelector}" />

            <Button Content="Save changes"

                    Grid.Row="1"

                    Grid.Column="0"

                    Margin="3"

                    Typography.Capitals="AllSmallCaps"

                    cal:Message.Attach="[Click]=[Save]" />

            <Button Content="Add new"

                    Grid.Row="1"

                    Grid.Column="1"

                    Margin="3"

                    Typography.Capitals="AllSmallCaps"

                    cal:Message.Attach="[Click]=[New]" />

            <Button Content="Delete"

                    Grid.Row="1"

                    Grid.Column="2"

                    Margin="3"

                    Typography.Capitals="AllSmallCaps"

                    cal:Message.Attach="[Click]=[Delete]" />

        </Grid>

    </Border>

</UserControl>

Ze względu na istniejące różnice między klasami poszczególnych kontrahentów, widok szczegółów posiada element ContentControl korzystający z klasy BusinessPartnerDataTemplateSelector do wstawienia odpowiedniego szablonu na podstawie typu powiązanej klasy.

public class BusinessPartnerDataTemplateSelector : DataTemplateSelector

    {

        public override DataTemplate SelectTemplate(object item, DependencyObject container)

        {

            switch (item)

            {

                case CustomerDto \_:

return Application.Current.FindResource("CustomerDataTemplate") as DataTemplate;

                case ShipperDto \_:

return Application.Current.FindResource("ShipperDataTemplate") as DataTemplate;

                case SupplierDto \_:

return Application.Current.FindResource("SupplierDataTemplate") as DataTemplate;

                default: return null;

            }

        }

    }

Szablony zostały zadeklarowane w pliku App.xaml:

            <DataTemplate x:Key="CustomerDataTemplate">

                <UniformGrid Columns="2">

                    <TextBlock Text="ID: " />

                    <TextBox Text="{Binding CustomerID}" />

                    <TextBlock Text="Company name: " />

                    <TextBox Text="{Binding CompanyName}" />

                    <TextBlock Text="Contact name: " />

                    <TextBox Text="{Binding ContactName}" />

                    <TextBlock Text="Contact title: " />

                    <TextBox Text="{Binding ContactTitle}" />

                    <TextBlock Text="Address: " />

                    <TextBox Text="{Binding Address}" />

                    <TextBlock Text="City: " />

                    <TextBox Text="{Binding City}" />

                    <TextBlock Text="Region: " />

                    <TextBox Text="{Binding Region}" />

                    <TextBlock Text="Postal code: " />

                    <TextBox Text="{Binding PostalCode}" />

                    <TextBlock Text="Country: " />

                    <TextBox Text="{Binding Country}" />

                    <TextBlock Text="Phone number: " />

                    <TextBox Text="{Binding Phone}" />

                    <TextBlock Text="Fax: " />

                    <TextBox Text="{Binding Fax}" />

                    <Grid />

                    <Grid/>

                </UniformGrid>

            </DataTemplate>

            <DataTemplate x:Key="SupplierDataTemplate">

                <UniformGrid Columns="2">

                    <TextBlock Text="ID: " />

                    <TextBox Text="{Binding SupplierID}" />

                    <TextBlock Text="Company name: " />

                    <TextBox Text="{Binding CompanyName}" />

                    <TextBlock Text="Contact name: " />

                    <TextBox Text="{Binding ContactName}" />

                    <TextBlock Text="Contact title: " />

                    <TextBox Text="{Binding ContactTitle}" />

                    <TextBlock Text="Address: " />

                    <TextBox Text="{Binding Address}" />

                    <TextBlock Text="City: " />

                    <TextBox Text="{Binding City}" />

                    <TextBlock Text="Region: " />

                    <TextBox Text="{Binding Region}" />

                    <TextBlock Text="Postal code: " />

                    <TextBox Text="{Binding PostalCode}" />

                    <TextBlock Text="Country: " />

                    <TextBox Text="{Binding Country}" />

                    <TextBlock Text="Phone number: " />

                    <TextBox Text="{Binding Phone}" />

                    <TextBlock Text="Fax: " />

                    <TextBox Text="{Binding Fax}" />

                    <TextBlock Text="Home page: " />

                    <TextBox Text="{Binding HomePage}" />

                </UniformGrid>

            </DataTemplate>

            <DataTemplate x:Key="ShipperDataTemplate">

                <UniformGrid Columns="2">

                    <TextBlock Text="ID: " />

                    <TextBox Text="{Binding ShipperID}" />

                    <TextBlock Text="Company name: " />

                    <TextBox Text="{Binding CompanyName}" />

                    <TextBlock Text="Phone number: " />

                    <TextBox Text="{Binding Phone}" />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

                    <Grid />

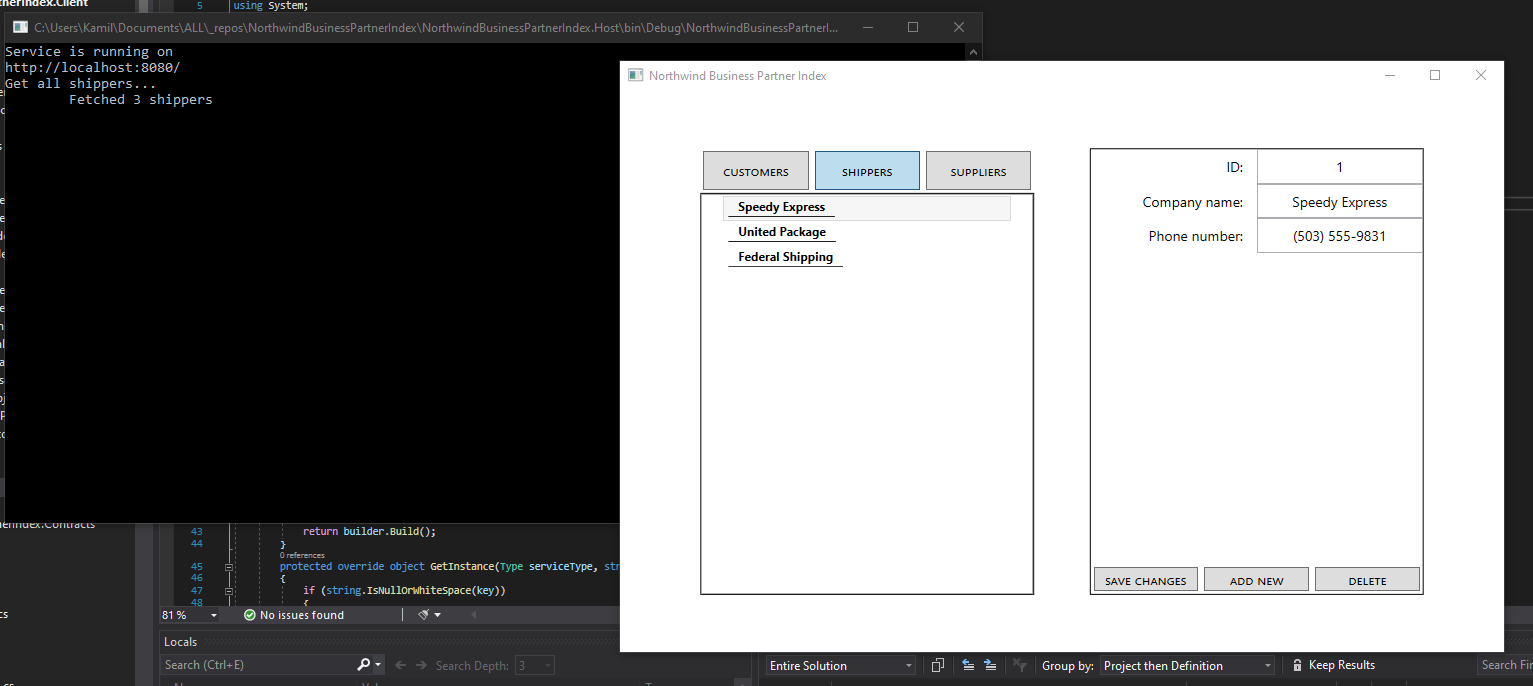
                    <Grid />

                </UniformGrid>

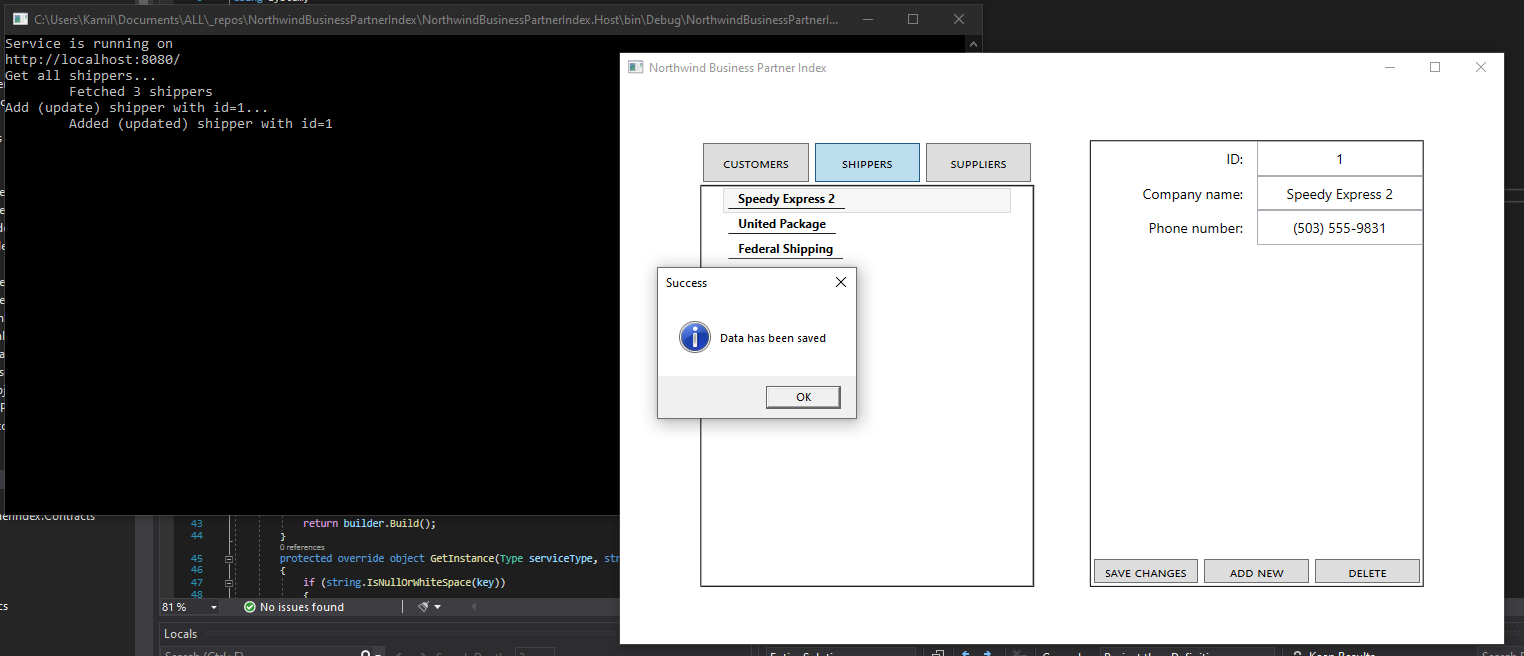
            </DataTemplate>

### Działanie programu

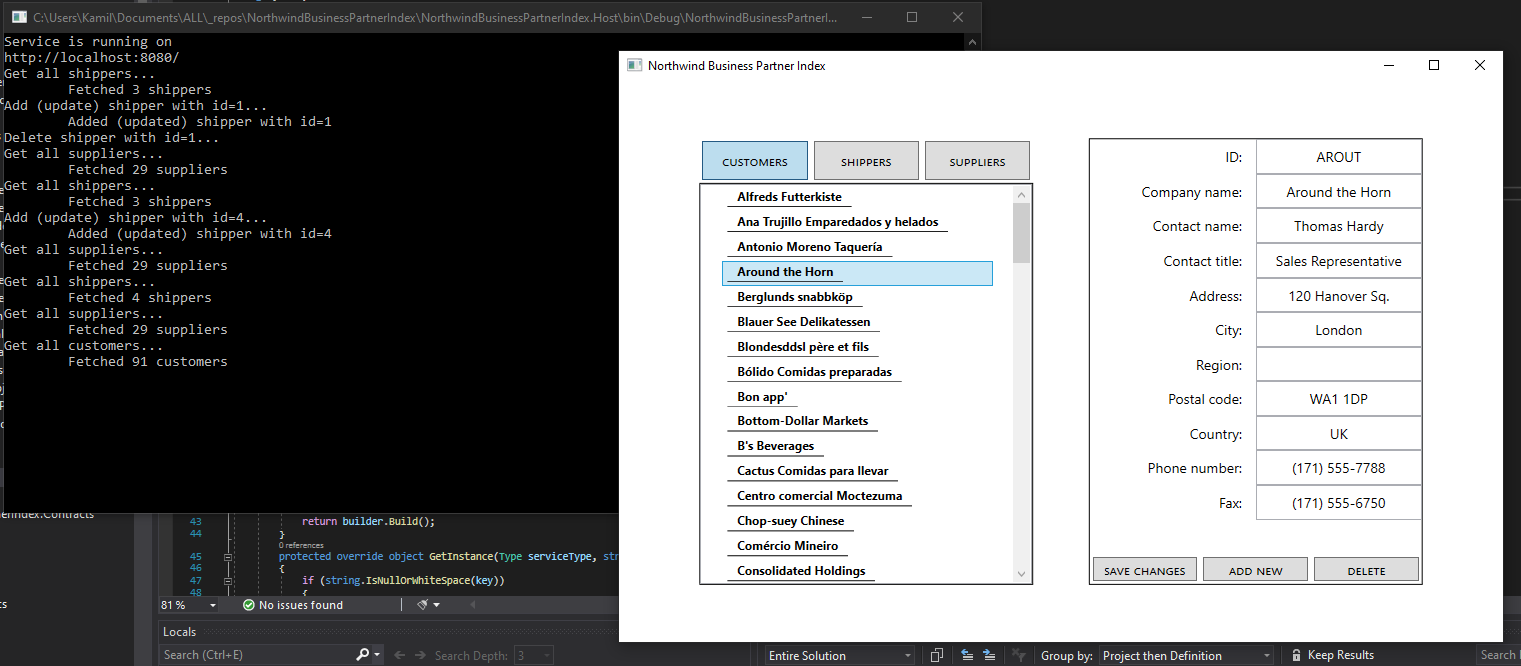
Pobranie listy przewoźników:



Edycja nazwy firmy jednego z przewoźników:



Lista klientów:



### Pliki projektu (klient)

