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

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

**Studia stacjonarne / niestacjonarne**

niestacjonarne

**Imię i nazwisko studenta: grupa:**

Kamil Nowak

**Temat projektu:**

Program udostępniający operacje CRUD dla tabeli 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 do wybranej tabeli.

## Serwer

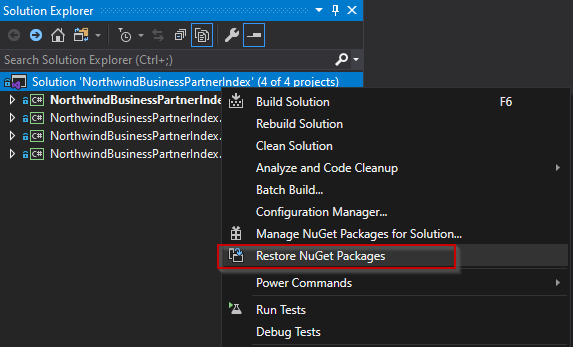
### Wymagania sprzętowe i systemowe oraz biblioteki

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 do pliku). 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/"/>

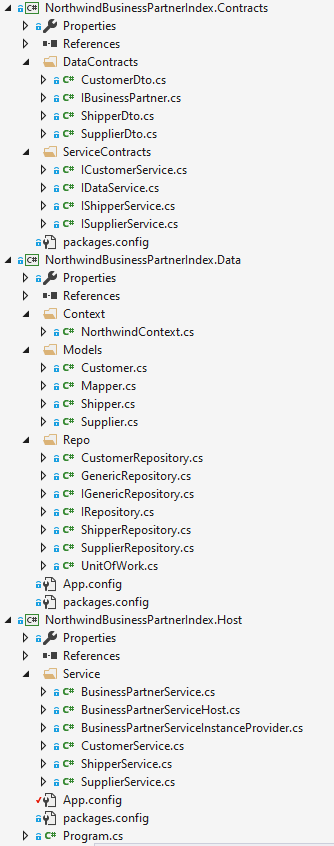
</baseAddresses>

</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
* Autofac (wersja 5.2) – wykorzystana do implementacji wzorca DI

Wszelkie zależności powinny zostać zaimportowane przy importowaniu zależności serwera w pkt. 2.1.

Tu opisać wymagania, jeśli są jakieś szczególne oraz biblioteki / narzędzia jeśli aplikacja klienta wymaga ich instalacji.

### Połączenie z serwerem i endpoint

Tu wkleić kluczowe fragmenty kodu klienta odpowiedzialne za połączenie z serwerem (w tym endpoint oraz użycie klasy ChannelFactory<T>). Kody wklejamy jako tekst (a nie zrzuty ekranu).

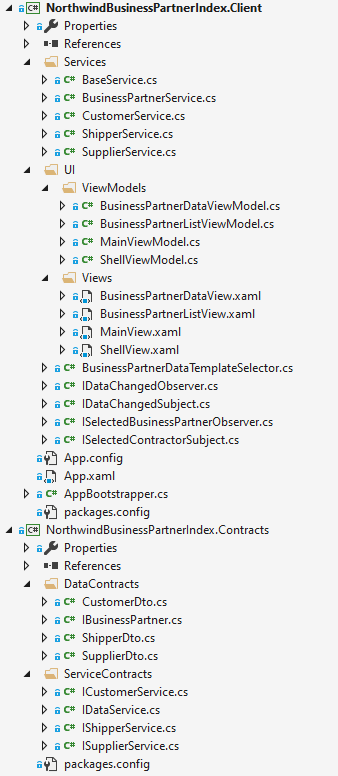
### Interfejs WPF – kod XAML

Tu wkleić cały kod XAML opisujący interfejs. Kody wklejamy jako tekst.

### Działanie programu

Tu wkleić kilka zrzutów ekranu interfejsu po uruchomieniu programu z krótkimi opisami.

### Pliki projektu (klient)



Tu można wkleić zrzut ekranu okna Solution Explorer (w Visual Studio). Jeżeli serwer i klient są w jednej solucji i zrzut został już umieszczony w punkcie 2.5, to tu wystarczy umieścić odnośnik do punktu 2.5.

## Pliki wgrane na Moodle

Na Moodle w opcji *Projekt zaliczeniowy (Program WCF) - studia stacjonarne i niestacjonarne* należy wgrać trzy pliki:

1. Plik zip (lub rar lub 7z) ze spakowanym projektem (serwer i klient w jednym archiwum). Plik ma mieć w nazwie nazwisko i imię (najpierw nazwisko), np. **Kowalski\_Jan.zip**.
2. Plik docx (lub doc) z opisem projektu, czyli niniejszy dokument. Plik ten ma mieć w nazwie nazwisko, imię studenta oraz skróconą nazwę projektu (skrócona nazwa ma zawierać jedno lub kilka kluczowych słów określających projekt). Przykładowa nazwa pliku z opisem: **Kowalski\_Jan\_Ekonometria.docx** (dla projektu z jakimiś obliczeniami z dziedziny ekonometrii).
3. Plik pdf z opisem projektu (zawartość taka sama jak w pliku docx) – plik ten należy przygotować na wypadek, jakby plik docx był w mojej wersji edytora źle sformatowany. Plik ma mieć tę samą nazwę co plik docx (i oczywiście rozszerzenie pdf).