Web service SOAP avec WCF et LINQ

Hyacinthe MENIET

3 août 2019

Les Web services SOAP sont un moyen d'interopérer avec des programmes écrits dans des langages différents ou qui s'exécutent sur des machines différentes. Ce document décrit les concepts et les étapes pour exposer sous forme de Web service SOAP des données SQL sérialisés via LINQ (Language Integrated Query). Dans ce chapitre, j'aborderai les notions de base dont vous aurez besoin pour comprendre et travailler avec les services WCF (Windows Communication Foundation). WCF étant l'implémentation par Microsoft d'un ensemble de normes qui définissent les interactions entre services, la (dé-)sérialisation d'objets et la gestion des protocoles associés.

1. Pré-requis

- Vous êtes familier de C# 7 (.NET Framework 4.6.2 minimum) et de sa syntaxe.
- Vous connaissez SOAP.
- Vous avez installé IIS (7.5 minimum) et sa console de gestion (`inetmgr`).
- Vous avez installé - à minima - et êtes familier de Microsoft SQL Server 2014 Express.
- Pour vous connecter à SQL Server 2014 Express vous utilisez l'Authentification SQL Server et non l'Authentification Windows.
- Vous avez installé et êtes familier de Microsoft Visual Studio (Community 2017 minimum). Les composants requis pour ce tutoriel sont **Développement .NET Desktop** et **Développement web et ASP.NET**

2. Vue d'ensemble

2.1 Présentation de WCF

WCF fournit aux développeurs l'essentiel pour développer, déployer, exécuter et consommer des services sous Windows. Cela inclus, l'hébergement, la gestion d'instance de service, les appels asynchrones, la fiabilité, la gestion des transactions, la gestion des files d'attente et la sécurité. WCF est un composant de Microsoft .NET.
WCF supporte plusieurs bindings :

Binding	Description	Transport	Codage	Inter ?
Basic binding	Ce binding est dédié aux clients et services basés sur ASMX,ou qui respectent le profil WS-I Basic Profile 1.1.	HTTP/HTTPS	Texte, MTOM	Oui
TCP binding	Binding sécurisée et fiable, optimisé pour les communications WCF-to-WCF.	TCP	Binaire	Non
IPC binding	Offre des fonctionnalités équivalentes à NetTcpBinding à ceci près qu'il est limité à la communication inter-processus (donc sur la même machine). C'est le plus performant.	IPC	Binaire	Non
Web Service (WS) binding	Ce binding est conçu pour interopérer avec tout framework prennant en charge la spécification WS-* (sessions fiables, transactions distribuées et sécurité).	HTTP/HTTPS	Texte, MTOM	Oui
Dual WS binding	Similaire au WSHttpBinding, de plus, il supporte les contrats de service duplex qui permettent aux services et clients d'envoyer et recevoir des messages.	HTTP	Texte, MTOM	Non
MSMQ binding	Ce binding est dédié aux communications avec file d'attente.	MSMQ	Binaire	Non

Ce tutoriel est consacré à SOAP, c'est pourquoi je me concentrerai sur *Web Service (WS) binding*. Dans WCF, tous les services exposent des contrats. Les contrats étant le moyen standard pour décrire ce que fait le service. WCF définit 4 types de contrats :

- **Contrat de services :** Décrit les opérations que le client peut effectuer via le service.
- **Contrat de données :** Définit de quels types sont les données qui sont passées depuis et vers le service. WCF définit des contrats implicites pour les types primitifs.
- **Contrat d'erreurs :** Précise les erreurs qui sont soulevées par le service et la façon dont le service gère les erreurs et les propage à ses clients.
- **Contrat de message :** Les contrats de message peuvent être dactylographiés ou non typé et sont utiles pour interopérer avec des services qui utilisent des formats propriétaires de message.

Enfin, chaque service WCF doit être hébergé dans un processus Windows appelé processus hôte. Un unique processus hôte peut héberger plusieurs services et le même service peut être hébergé dans de multiples processus hôtes. WCF supporte les processus hôtes suivants :

- **Internet Information Services (IIS) 5/6 :** Limités aux services transportés via HTTP(S).
- **Self-hosting (Auto-hébergement) :** Dans ce cas, le processus hôte est une application Windows Forms, une application console, ou un service NT. Ce mode d'hébergement supporte tous les types de transport pris en charge par WCF.
- **Internet Information Services (IIS) 7 :** Il supporte tous les types de transport pris en charge par WCF.

2.2 Présentation rapide de LINQ

LINQ est une technologie livrée avec Microsoft .NET. LINQ permet d'interroger des sources de données directement en C#. Traditionnellement, pour interroger des données structurées le programmeur devait passer par un langage externe comme SQL ou XPath. Avec LINQ cette étape n'est plus nécessaire.
LINQ supporte plusieurs type de sources de données, notamment les bases de données SQL, les documents XML, les DataSet ADO.NET et les collections d'objets en mémoire. LINQ to SQL fait référence aux capacités ORM (object-relational mapping) de LINQ.

2.3 Présentation de l'article

Pour rendre ce tutoriel digeste, je vais parcourir les capacités de WCF et LINQ à travers l'exemple d'un logiciel de gestion de finances personnelles de type Microsoft Money. Ce logiciel sera multi-comptes et utilisable à distance via son API Web services SOAP. Les 6 opérations exposées sont les suivantes :

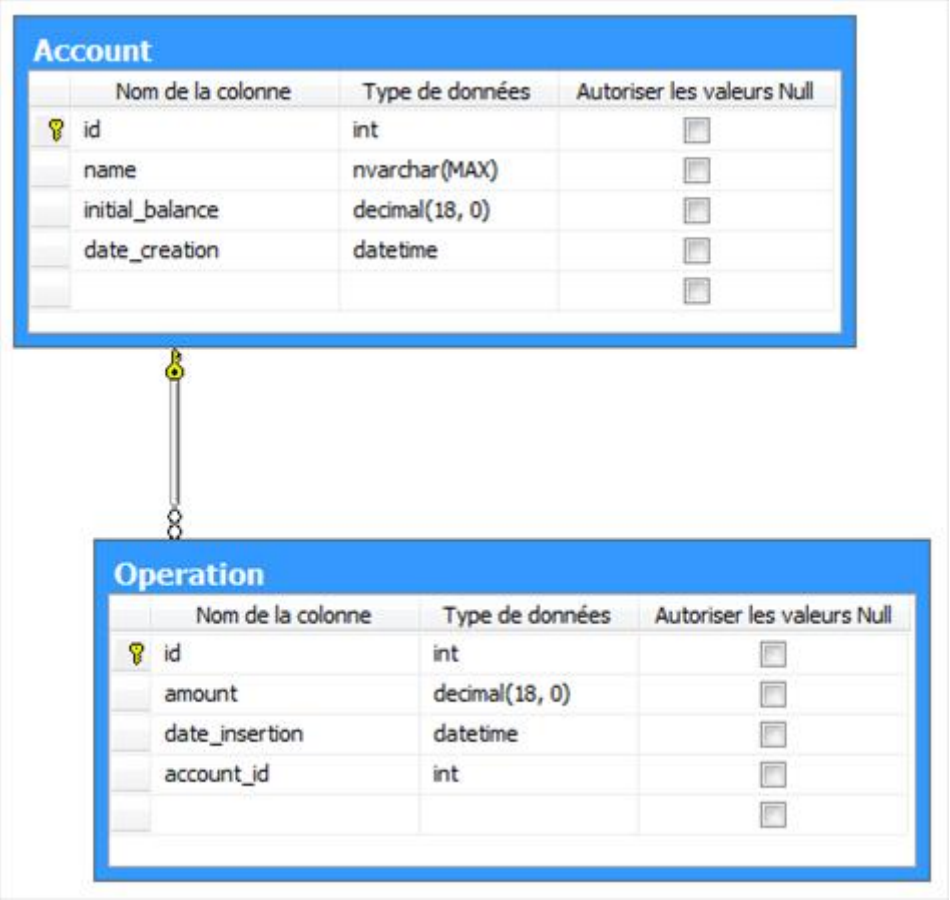
- Lister les comptes
- Lister les opérations sur un compte.
- Récupérer le solde d'un compte.
- Retirer de l'argent d'un compte.
- Verser de l'argent sur un compte.
- Transférer de l'argent d'un compte vers un autre.

Vous pouvez télécharger le code source du projet WCF et LINQ pour Visual Studio Community 2017.

3. Le Web service

3.1 La base de données

Créez la base de données **Banking** contenant les 2 tables Operation et Account suivantes :



J'ai réalisé ce schéma avec Microsoft SQL Server Management Studio 2014. Cette capture correspond au schéma de données suivant :

```
USE [Banking]
GO
/****** Object:  Table [dbo].[Account]    Script Date: 27/12/2017
    11:53:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Account](
[id] [int] IDENTITY(1,1) NOT NULL,
[name] [nvarchar](max) NOT NULL,
[initial_balance] [decimal](18, 0) NOT NULL,
[date_creation] [datetime] NOT NULL,
CONSTRAINT [PK_Account] PRIMARY KEY CLUSTERED
(
[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
    ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

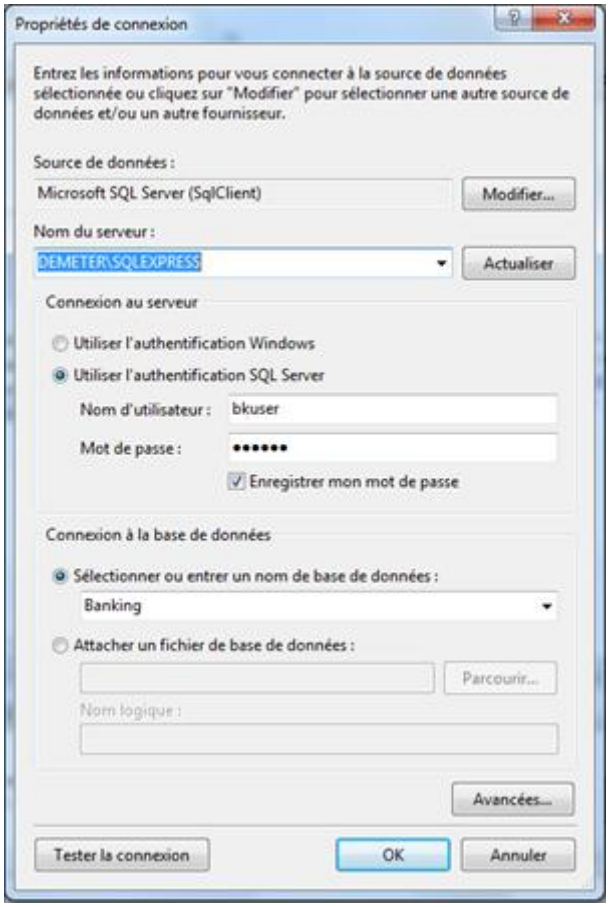
```
GO
/****** Object:  Table [dbo].[Operation]    Script Date:
    27/12/2017 11:53:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Operation](
[id] [int] IDENTITY(1,1) NOT NULL,
[amount] [decimal](18, 0) NOT NULL,
[date_insertion] [datetime] NOT NULL,
[account_id] [int] NOT NULL,
CONSTRAINT [PK_Operation] PRIMARY KEY CLUSTERED
(
[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
     ON) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Operation]  WITH CHECK ADD  CONSTRAINT [
    FK_Operation_Account] FOREIGN KEY([account_id])
REFERENCES [dbo].[Account] ([id])
GO
ALTER TABLE [dbo].[Operation] CHECK CONSTRAINT [
    FK_Operation_Account]
GO
```

Initialisez la table Account avec les données ci-dessous :

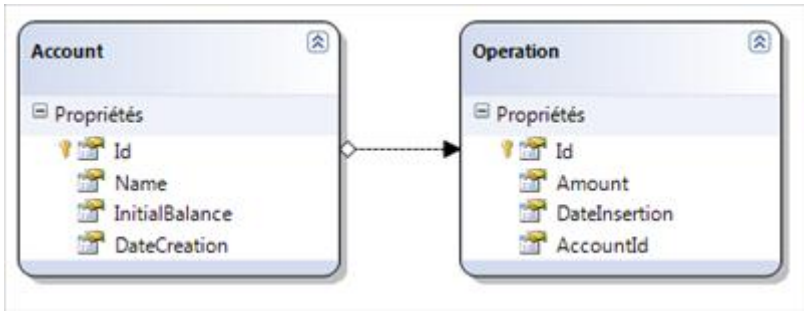
id	name	initial_balance	date_creation
1	Banque Populaire	50	2010-02-04 00:00:00.000
2	Societe Generale	120	2010-02-04 00:00:00.000
3	BNP Paribas	70	2010-02-03 00:00:00.000

3.2 La couche persistance

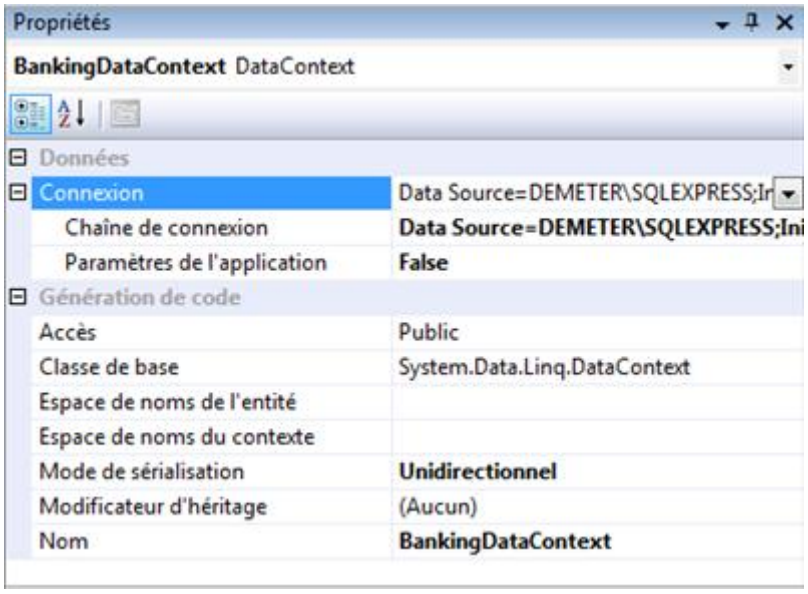
Dans Visual Studio, créez un projet *Application du service WCF* nommée **Dotmyself**. Ajoutez un nouvel élément de type *Classes LINQ to SQL* nommé **Banking.dbml** à votre projet. Cela ouvre l'outil Concepteur Objet/Relationnel. Depuis l'Explorateur de bases de données, connectez-vous à la base de données Banking :



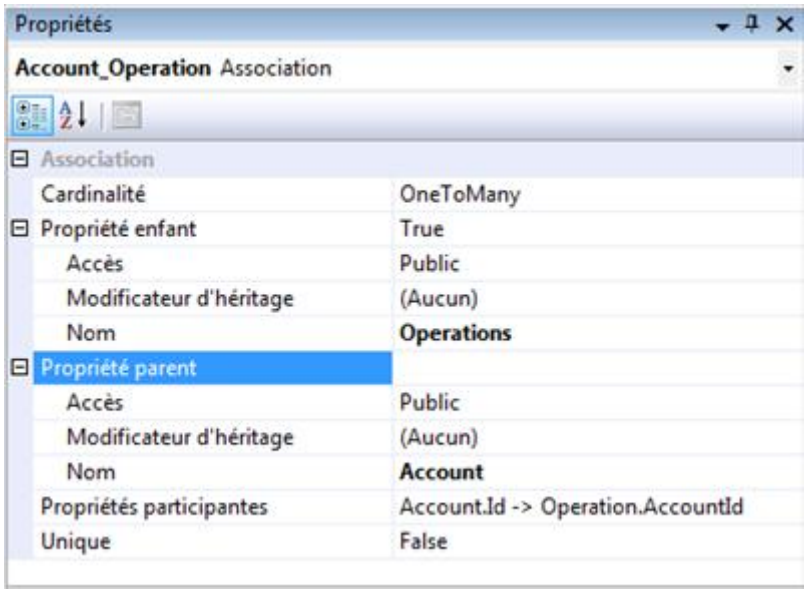
M19555 est le nom du serveur sur lequel est installé mon SQL Server.
Faîtes glisser déposer, dans l'outil Concepteur Objet/Relationnel, les tables Operation et Account :



Assurez-vous que votre BankingDataContext ressemble à ceci :



Enfin, éditez l'association entre Operation et Account, là ajoutez un *« s »* à Operation s'il n'en a pas



3.3 Encapsulation de la couche persistance

Créez la classe DataManager.cs qui offre un accès générique à la couche persistance :

```
using System;
using System.Collections.Generic;
using System.Data.Linq.Mapping;
using System.Linq;
using System.Linq.Expressions;
namespace Dotmyself
{
/// <summary>
```

```
/// It offers a generic CRUD abstraction for entities.
/// Author : Hyacinthe MENIET
/// </summary>
/// <typeparam name="T">
/// Entity that a client specifies when it instantiates this
    class.
/// </typeparam>
public class DataManager<T> where T : class
{
private readonly BankingDataContext context;
public DataManager(BankingDataContext dc)
{
context = dc;
}
public virtual IEnumerable<T> List() => context.GetTable<T>();
public virtual T Get(int id)
{
MetaTable mapping = context.Mapping.GetTable(typeof(T));
MetaDataMember pkfield = mapping.RowType.DataMembers.
    SingleOrDefault(
d => d.IsPrimaryKey);
ParameterExpression param = Expression.Parameter(typeof(T), "e");
var p = Expression.Lambda<Func<T, bool>>(
Expression.Equal(Expression.Property(param, pkfield.Name),
Expression.Constant(id)),
new ParameterExpression[] { param });
return context.GetTable<T>().SingleOrDefault(p);
}
public virtual void Create(T entity) => context.GetTable<T>().
    InsertOnSubmit(entity);
public virtual void Update(T entity) => context.GetTable<T>().
    Attach(entity);
public virtual void Delete(T entity) => context.GetTable<T>().
    DeleteOnSubmit(entity);
public virtual void Commit() => context.SubmitChanges();
}
}
```

3.4 La couche service

Créez le contrat de service IBankingService.cs :

```
using System.Collections.Generic;
using System.ServiceModel;
namespace Dotmyself
{
/// <summary>
/// It defines the capability and feature set, offered by the
```

```
    Banking Service.
/// Author : Hyacinthe MENIET
/// </summary>
[ServiceContract(Namespace = "http://file.dotmyself.net/source
    /10")]
interface IBankingService
{
[OperationContract]
IList<Account> GetAccounts();
[OperationContract]
IList<Operation> GetOperationsByAccount(int aId);
[OperationContract]
decimal GetBalance(int aId);
[OperationContract]
void withdrawMoney(int aId, decimal amount);
[OperationContract]
void depositMoney(int aId, decimal amount);
[OperationContract]
void transferMoney(int fromId, int toId, decimal amount);
}
}
```

Puis son implémentation BankingService.svc.cs :

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
namespace Dotmyself
{
/// <summary>
/// It implements the operations the client can perform on the
    Banking Service.
/// Author : Hyacinthe MENIET
/// </summary>
public class BankingService : IBankingService
{
private readonly DataManager<Account> accountManager;
private readonly DataManager<Operation> operationManager;
public BankingService()
{
string ConnectStr = ConfigurationManager.ConnectionStrings["
    BankingConnectionString"].ConnectionString;
BankingDataContext context = new BankingDataContext(ConnectStr);
accountManager = new DataManager<Account>(context);
operationManager = new DataManager<Operation>(context);
}
public void depositMoney(int aId, decimal amount)
{
```

```
Operation op = new Operation
{
amount = Math.Abs(amount),
account_id = aId,
date_insertion = DateTime.Now
};
operationManager.Create(op);
operationManager.Commit();
}
public IList<Account> GetAccounts()
{
return accountManager.List().ToList();
}
public decimal GetBalance(int aId)
{
Account ac = accountManager.Get(aId);
decimal sumOp = ac.Operations.Sum(o => o.amount);
return sumOp + ac.initial_balance;
}
public IList<Operation> GetOperationsByAccount(int aId)
{
Account ac = accountManager.Get(aId);
return ac.Operations.ToList();
}
public void transferMoney(int fromId, int toId, decimal amount)
{
Operation opFrom = new Operation
{
amount = -1 * Math.Abs(amount),
account_id = fromId,
date_insertion = DateTime.Now
};
Operation opTo = new Operation
{
amount = Math.Abs(amount),
account_id = toId,
date_insertion = DateTime.Now
};
operationManager.Create(opFrom);
operationManager.Create(opTo);
operationManager.Commit();
}
public void withdrawMoney(int aId, decimal amount)
{
Operation op = new Operation
{
amount = -1 * Math.Abs(amount),
account_id = aId,
date_insertion = DateTime.Now
```

```
};
operationManager.Create(op);
operationManager.Commit();
}
}
}
```

Terminez par le fichier de service BankingService.svc :

```
<%@ ServiceHost Language="C#" Debug="true" Service="Dotmyself.
    BankingService" CodeBehind="BankingService.svc.cs" %>
```

4. Déploiement du Web service sur IIS

4.1 Fichier de configuration

Ecrasez votre Web.config par celui-ci :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<connectionStrings>
<add name="BankingConnectionString" connectionString="Data Source
    =M19555\SQLEXPRESS;Initial Catalog=Banking;Persist Security
    Info=True;User ID=bkuser;Password=dec5apR6thEj"
providerName="System.Data.SqlClient" />
</connectionStrings>
<system.serviceModel>
<services>
<service behaviorConfiguration="Dotmyself.BankingBehavior" name="
    Dotmyself.BankingService">
<endpoint address="" binding="wsHttpBinding" contract="Dotmyself.
    IBankingService" />
<endpoint address="mex" binding="mexHttpBinding" contract="
    IMetadataExchange" />
</service>
</services>
<behaviors>
<serviceBehaviors>
<behavior name="Dotmyself.BankingBehavior">
<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
<serviceDebug includeExceptionDetailInFaults="true"/>
</behavior>
</serviceBehaviors>
</behaviors>
</system.serviceModel>
</configuration>
```

Adaptez la **connectionString** à ce qu'il y a chez vous.

4.2 Configuration d'IIS

Pour fonctionner correctement dans IIS 7 (et plus), une application WCF doit utiliser un pool d'applications dont le mode pipeline est *« intégré »* . Je suppose dans la suite que IIS et son gestionnaire de services Internet (`inetmgr`) sont installés. Lancez un terminal Windows (`cmd`) en tant qu'administrateur puis tapez les commandes suivantes :

```
C:textbackslash Windowstextbackslash Microsoft.NETtextbackslash
    Framework64textbackslash v3.0textbackslash Windows
    Communication Foundationtextbackslash ServiceModelReg.exe -i
C:textbackslash Windowstextbackslash Microsoft.NETtextbackslash
    Framework64textbackslash v4.0.30319textbackslash aspnet_regiis
    -i
```

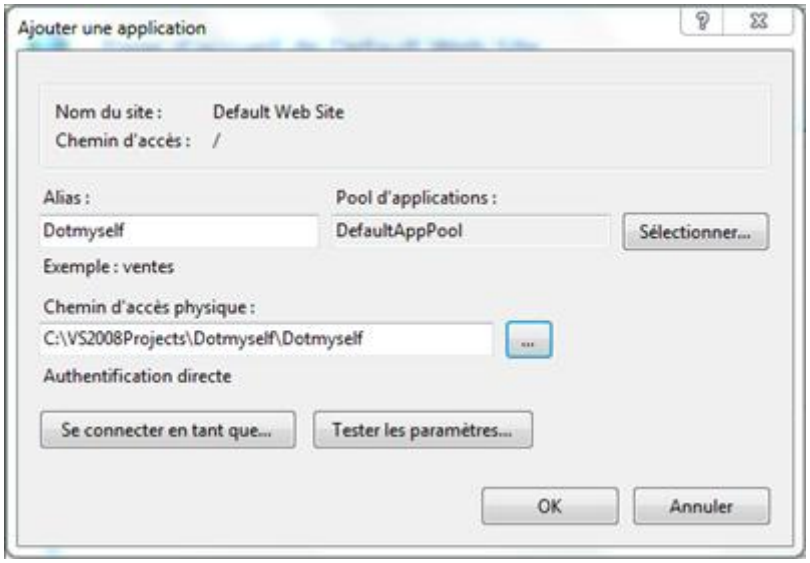
Allez dans *« Panneau de Configuration > Programmes > Activer ou désactiver des fonctionnalités Windows »*, dans le menu *« Microsoft .NET Framework 3.5 »*, activez :

1. Windows Communication Foundation HTTP Activation
2. Windows Communication Foundation Non-HTTP Activation

Validez et fermez. Redémarrez votre Windows pour être sûr que ces changements sont correctement pris en compte.

4.3 Déploiement dans IIS

Générez (Build) votre projet **Dotmyself**. Lancez le Gestionnaire de services Internet (`inetmgr`) et là ajoutez une nouvelle application au *Default Web Site* :



Dans Chemin d'accès physique prenez soin d'indiquer le dossier qui contient `BankingService.svc` Pour valider que l'application est déployée avec succès, vérifiez que la page des métadonnées s'affiche correctement : http://localhost/Dotmyself/BankingService.svc. De plus, vous obtiendrez le WSDL à l'adresse suivante : http://localhost/Dotmyself/BankingService.svc?wsdl

5. Le client

5.1 Ecriture du client

Lancez Microsoft Visual Studio et créez un projet *Application console* nommé **Client**. Cliquez droit sur Reference et ajoutez une référence de service depuis l'URL http://localhost/Dotmyself/BankingService.svc Cliquez sur **Allez** et remplacez l'espace de nom par **Dotmyself**. Validez et fermez.
Modifiez la classe Program.cs ainsi :

```
using Client.Dotmyself;
using System;
namespace Client
{
/// <summary>
/// It invokes operations on the Banking Service.
/// Author : Hyacinthe MENIET
/// </summary>
class Program
{
static void Main(string[] args)
{
BankingServiceClient client = new BankingServiceClient();
Console.WriteLine("List all accounts :");
foreach (Account ac in client.GetAccounts())
{
Console.WriteLine("{Id=" + ac.id + ",Name=" + ac.name
+ ",InitialBalance=" + ac.initial_balance + ",DateCreation="
+ ac.date_creation + "}");
}
Console.WriteLine("Deposit 20 euros in account 1");
client.depositMoney(1, 20);
Console.WriteLine("Withdraw 10 euros in account 1");
client.withdrawMoney(1, 10);
Console.WriteLine("Transfer 30 euros from account 3 to account
    1");
client.transferMoney(3, 1, 30);
Console.WriteLine("List operations on account 1 :");
foreach (Operation op in client.GetOperationsByAccount(1))
{
Console.WriteLine("{Id=" + op.id + ",Amount=" + op.amount
+ ",DateInsertion=" + op.date_insertion + "}");
}
Console.WriteLine("Account 1 balance : " + client.GetBalance(1));
Console.WriteLine("Account 3 balance : " + client.GetBalance(3));
Console.ReadLine();
client.Close();
}
}
}
```

5.2 Exécution

Compilez le client et exécutez-le, vous devriez obtenir le résultat suivant :

```
List all accounts :
{Id=1,Name=Banque Populaire,InitialBalance=50,DateCreation
    =27/12/2017 13:27:48}
{Id=2,Name=Societe Generale,InitialBalance=120,DateCreation
    =27/12/2017 13:27:48}
{Id=3,Name=BNP Paribas,InitialBalance=70,DateCreation=27/12/2017
    13:27:48}
Deposit 20 euros in account 1
Withdraw 10 euros in account 1
Transfer 30 euros from account 3 to account 1
List operations on account 1 :
{Id=1,Amount=20,DateInsertion=27/12/2017 16:03:13}
{Id=2,Amount=-10,DateInsertion=27/12/2017 16:03:13}
{Id=4,Amount=30,DateInsertion=27/12/2017 16:03:13}
Account 1 balance : 90
Account 3 balance : 40
```