WCF RESTful JSON

1 Préambule

WCF basé sur le modèle générique A,B,C, il permet de développer des Web Services de tout type tant RESTFull que WS-SOAP.
Nous allons voir dans une série de TP comment nous pouvons substituer facilement différentes technologies dans la programmation d'un même web service, offrant souvent de nombreux Endpoints Cet article explique les pour vous, alors vous pouvez vous concentrer sur la logique métier plutôt que configuration de vos services WCF.

2 Un premier service WCF RESTFul JSON

Le projet à créer sous Visual Studio sera ici un Projet C# / WCF / Bibliothèque du service WCF.
Un code source est déjà généré dans les trois fichiers majeurs du projet :

- AppConfig pour la description des "Bindings" de votre service
- IService1.cs pour la description de l'interface de votre service
- Service1.cs pour la description du corps de votre service

2.1 Bibliothèque System.ServiceModel.Web :

Ensuite, nous devons ajouter une référence pour l'espace de nom System.ServiceModel.Web (Ajouter une référence au projet). Cette bibliothèque fournit des classes pour la mise en œuvre des services Web, comme WebInvoke que nous verrons plus loin.
A noter qu'il vous faudra le .Net Framework 4 ou plus récent. Si vous ne l'avez pas installé, cela devrait vous être proposé.

2.2 Fichier App.Config

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
       <services>
         <service name="WcfJsonRestService.Service1">
              <endpoint address="http://localhost:8732/service1"
                              binding="webHttpBinding"
                              contract="WcfJsonRestService.IService1"/>
         </service>
       </services>
       <behaviors>
         <endpointBehaviors>
              <behavior>
                <webHttp />
              </behavior>
         </endpointBehaviors>
       </behaviors>
  </system.serviceModel>
</configuration>
```

Par rapport au projet initialement généré par visual studio nous avons remplacé le binding wsHttpBinding par webHttpBinding.

- Quelle est la modification qui en découle ?
- L'autre changement important est l'addition d'un endpointBehavior pour webHttp. A quoi cela sert-il ?
- Pourquoi ces deux changements sont nécessaires pour permettre la mise en place d'un service REST / JSON avec WCF ?

2.3 Fichier IService1:

```
using System.ServiceModel;

namespace WcfJsonRestService
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        Person GetData(string id);
    }
}
```

Notez la présence d'un paramètre "id" de type string dans l'appel de la méthode GetData.

Dans cet exemple, nous retournons une donnée de type "Person".

```
using System;
using System.ServiceModel.Web;

namespace WcfJsonRestService
{
    public class Service1 : IService1
    {
        [WebInvoke(Method = "GET",
                   ResponseFormat = WebMessageFormat.Json,
                   UriTemplate = "data/{id}")]
        public Person GetData(string id)
        {
            // lookup person with the requested id
            return new Person()
                       {
                           Id = Convert.ToInt32(id),
                           Name = "Leo Messi"
                       };
        }
    }

    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

- Quelles sont les méthodes utilisées par Service1.cs appartenant à la bibliothèque System.ServiceModel ?

3 Test du Service Web RESTFul

Pour analyser les échanges protocolaires sous-jacents nous allons utiliser un browser Web comme Firefox.

Conformément au modèle REST, les requêtes du Web Service utilisent des URI.

Un EndPoint REST prend la forme : `http://server/site/_api/lists/getbytitle('listname')/items`

- Nous testons donc notre service au travers l'URI : http://localhost:8732/Service1/data/10
- Puis nous irons analyser les échanges entre le client Firefox et le Service grâce aux outils/développeur Web/Inspecteur de Firefox.
- Quelles sont les messages HTTP de requête et de réponses.
- Qu'elle est l'intérêt d'utiliser un tel framework WCF ?

ATTENTION : La gestion des droits d'accès et de réservation des URL dans Visual Studio, au lancement du serveur HTTP peuvent provoquer une erreur sur l'installation du service à l'adresse http://localhost:8732/service1. La gestion de ces droits d'accès se fait au travers la commande netsh de gestion des interfaces et services réseau de votre machine en général.
Dans notre cas il faudra utiliser la commande suivante pour utiliser l'URL du projet :

4 Utilisation de REST dans les Architectures orientées Ressources (ROA)

4.1 CRUD :

CRUD (pour Create, Read, Update, Delete) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

REST est un donc un bon modèle et REST/JSON une implémentation possible de service dans une Architecture orientée Ressources à partir du tableau de correspondance suivant :

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE

4.2 WCF RESTFul JSON CRUD :

Sur la base du service WCF RESTFul JSON implémenté dans la première partie, nous allons développer un service avec les fonctionnalités suivantes :

- La possibilité de rajouter ou updater une donnée de type Person dans un cache (nous utiliserons la Classe DataTable du .Net Framework. Cf. section 4.3) sauvegardable dans un fichier.
- La possibilité d'effacer une donnée de type Person

4.3 Utilitaire cURL de Test :

Nous allons tester votre travail avec l'utilitaire en ligne de commande Curl qui permet d'accéder à toutes les commandes HTTP et idéal pour tester les API REST.

Vous pouvez le télécharger sur http://curl.haxx.se/download.html, en particulier pour Windows.

Voici quelques exemples d'utilisation de cURL :

Pour déposer un document, envoyer un PUT au EndPoint avec l'URI du document comme paramètre. Une extension de fichier .xml, .json, ou .txt indique son type. Vous pouvez remplacer ce dernier par un paramètre contentType.
Par exemple, nous déposons ici un document au format JSON à l'URI /afternoon-drink.json :

```
curl -X PUT -d '{"name": "Iced Mocha", "size": "Grandé", "tasty": true }' 'http://myhost/store?uri=/afternoon-drink.json'
```

Pour obtenir un document à partir de son URI, vous pouvez utiliser une requête GET:

```
curl -X GET 'http://myhost/store?uri=/afternoon-drink.json'
```

Pour mettre à jour un document, vous pouvez envoyer une requête POST :

```
curl -X POST -d '{"name": "Iced Tea", "size": "Jumbo", "refreshing": true }' 'http://myhost/store?uri=/afternoon-drink.json'
```

Pour supprimer un document, vous pouvez envoyer une requête DELETE :

```
curl -X DELETE 'http://myhost/store?uri=/afternoon-drink.json'
```

4.4 Sauvegarde et Restauration dans des fichiers CSV

Ce TP est l'occasion d'introduire la Classe DataTable : http://msdn.microsoft.com/fr-fr/library/system.data.datatable. Cette classe permet le stockage de données dans un cache.

WCF RESTful JSON

Voici un exemple de code qui permet de charger un fichier CSV dans un objet DataTable.

```
private static DataTable GetDataTabletFromCSVFile(string csv_file_path)
        {
            DataTable csvData = new DataTable();
            try
            {
              using(TextFieldParser csvReader = new TextFieldParser(csv_file_path))
                 {
                    csvReader.SetDelimiters(new string[] { "," });
                    csvReader.HasFieldsEnclosedInQuotes = true;
                    string[] colFields = csvReader.ReadFields();
                    foreach (string column in colFields)
                    {
                        DataColumn datecolumn = new DataColumn(column);
                        datecolumn.AllowDBNull = true;
                        csvData.Columns.Add(datecolumn);
                    }
                    while (!csvReader.EndOfData)
                    {
                        string[] fieldData = csvReader.ReadFields();
                        //Making empty value as null
                        for (int i = 0; i < fieldData.Length; i++)
                        {
                            if (fieldData[i] == "")
                            {
                                fieldData[i] = null;
                            }
                        }
```