Services Web de type REST
Développement sous C# .Net

1 Introduction

1.1 Bref rappel du Cours « du Web au Web services » (SI3)

Les services web de type REST sont les plus simples et les plus intuitifs qui soient pour peu que l'on ait pratiqué la gestion de pages web dynamiques avec un serveur Web et le protocole HTTP.
En effet, dans le cas de pages web dynamiques les données de réponse à un GET ou à un POST, sont générées à la volée côté serveur (par un cgi-bin par exemple). Elles sont destinées à une famille de clients bien définies : les browsers web (ex. Chrome, Firefox, IExplorer, …). Les formats des données renvoyées par le serveur sont donc contraintes par les capacité des clients à les lire, parser et afficher. Dans le cas des services Web, les clients peuvent être toute sorte d'applications et sont seulement contraintes par le protocole d'échange utilisé pour communiquer avec le serveur Web (souvent du JSON, de l'XML ou du CSV). C'est ainsi que les Services Web peuvent se retrouver au cœur du développement d'applications réparties.

1.2 Les 5 principes REST

REST (Representational State Transfer) ou RESTful est donc un style d'architecture permettant de construire des applications réparties. Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière. L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche (comme nous le verrons avec les web services WS-SOAP).

Principe n°1 : l'URI est l'identifiant des ressources

Principe n°2 : les verbes HTTP comme opérations sur les ressources :

- Créer (create) => POST
- Afficher (read) => GET
- Mettre à jour (update) => PUT
- Supprimer (delete) => DELETE

Principe n°3 : les réponses HTTP comme représentation des ressources (contenu des ressources codé en JSON, XML, CSV ou autre)

Et parfois les principes n°4 et n°5 avec des liens comme relation entre les ressources et un paramètre comme jeton d'authentification pour les accès contrôlé à certain services web (ex. Google API). Préambule

2 Modèle ABC et WCF .Net et un premier service WCF RESTFul JSON

WCF basé sur le modèle générique « A B C » permet de développer des Web Services de tout type dont RESTFul.

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 » (point de terminaison pour une connexion à un service avec une adresse IP, un port, un protocole d'échange…)

question 1 : Créer sous Visual Studio un Projet C# / WCF / Bibliothèque du service WCF.

Un code source est déjà généré dans les trois fichiers majeurs du projet :

- IService1.cs pour la description de l'interface de votre service
- Service1.cs pour la description du corps de votre service
- App.config pour la description des "Bindings" de votre service, i.e. tous les éléments de configuration protocolaire de vos services (URI, IP, port, protocole d'échange, protocole d'encodage etc.)

NOTES sur App.Config : Les fichiers de configuration sont des fichiers XML qui contiennent la configuration de notre exécutable. Cela peut être des chaines de connexions, des valeurs paramétrables, une url de web services, des préférences utilisateurs, des contrôles personnalisés etc. Ces fichiers doivent se situer dans le même répertoire que l'exécutable. Pourquoi utiliser un fichier de configuration ?

- pour éviter de mettre des valeurs en dur dans le code. Imaginons que nous utilisions une url de web service dans notre application, si l'url change, on aimerait ne pas avoir à recompiler le code.
- pour éviter d'utiliser la base de registre et de donner les droits de modification de base de registre.

Ces fichiers permettent de typer les données à sauvegarder et le framework .Net dispose de méthodes pour y accéder facilement.
L'intérêt d'utiliser un fichier XML plutôt qu'un fichier binaire est que ce fichier est lisible et compréhensible facilement. On peut également le modifier à la main sans un système évolué permettant de faire des modifications.

2.1 Bibliothèque System.ServiceModel.Web :

ATTENTION : Si certaines libraires et donc espaces de noms ne sont pas trouvés (comme par exemple) System.ServiceModel.Web, vous devez les chargés dans votre projet avec ajouter références/Assemblys/Framework.

Ainsi, 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.

Le fichier App.Config du projet de type WCF permet de configurer principalement la mise en œuvre des classes de System.ServiceModel.Web utilisées pour la conception du service.
En voici un très simple pour notre projet en cours.

```
<?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.

question 2 : Quelle est la modification qui en découle d'après vous ? Vous pouvez faire une recherche sur le web et MSDN pour trouver cette information …

question 3 : L'autre changement important est l'addition d'un endpointBehavior pour webHttp. A quoi cela sert-il ?

question 4 : Pourquoi ces deux changements sont nécessaires pour permettre la mise en place d'un service REST / JSON avec WCF ?

2.2 Fichier IService1:

```
using System.ServiceModel;

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

Les annotations [ServiceContract] et [OperationContract] servent à marquer les interfaces et les méthodes qui feront l'objet d'une exportation au travers le service web.
Notez la présence d'un paramètre "id" de type string dans l'appel de la méthode GetData. Il s'agit du passage de paramètre que l'on retrouvera dans l'URI.

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; }
    }
```

L'annotation [WebInvoke (…)] défini les paramètres de configuration pour l'accès à la mathode

question 5 : Quelles sont les méthodes utilisées par Service1.cs appartenant à la bibliothèque System.ServiceModel.Web ?

3 Test du Service Web RESTFul

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

question 6 : Pourquoi pouvons-nous utiliser un browser Web pour tester ce service ?

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

question 7 : Nous testons donc notre service au travers l'URI : http://localhost:8732/Service1/data/10

question 8 : Puis nous irons analyser les échanges entre le client Firefox et le Service grâce aux outils/développeur Web/Inspecteur de Firefox.

question 9 : Quels sont les messages HTTP de requête et de réponses.

question 10 : Quel est l'intérêt d'utiliser un tel framework WCF ?

4 Optionnel : Suivi de la disponibilité des Vélibs

question 11 : Pour les étudiants les plus avancés écrivez un client sui stockera l'évolution de la disponibilité des Vélibs dans un fichier CSV éditable sous excel …

Pour vous aider : La sauvegarde et restauration des données dans des fichiers CSV peut se faire au travers Classe DataTable : http://msdn.microsoft.com/fr-fr/library/system.data.datatable. Cette classe permet le stockage de données dans un cache.

Voici un exemple de code qui permet de charger

```
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;
                            }
                        }
```