TP5 Service Oriented Architecture

Ce TP est une introduction au SOA et plus particulièrement aux services Web et aux types interfaces qu'ils peuvent exposer. Nous verrons plus en détails comment implémenter et orchestrer des services Web dans le module Programmation de Web services. Dans ce TP, le focus est **(i) sur l'importance des interfaces dans les services et (ii) sur la différence entre APIs de type RPC vs Ressources**.

RPC-oriented APIs

Dans le cours nous avons vu les différents styles d'interfaces qu'un service peut offrir. Les Web services SOAP sont une implémentation des services utilisant le style remote procedure call (RPC). Dans cet exercice, nous allons lancer des requêtes au Web service SOAP suivant : http://www.dneonline.com/calculator.asmx. Il s'agit d'un service permettant de faire des calculs simples ….

Pour faire notre requête, nous utiliserons l'outil suivant :
http://www.soapclient.com/soaptest.html

Nous souhaitons maintenant faire une requête pour obtenir le résultat de la division de 2340 par 9.

Lorsque vous utilisez http://www.soapclient.com/soaptest.html,

- **Quelle information vous est demandée en premier ?**
- **Comment l'obtenez-vous ?**
- **Pourquoi ?**
- **Que se passe t'il lorsque vous avez donné cette information ?**

Génération d'une interface REST

L'objectif de cet exercice est (i) que vous voyez que l'API d'un service peut être facilement générée à partir du moment où l'on s'est mis d'accord sur les protocoles utilisés pour construire cette interface et (ii) que vous voyez la différence entre service SOAP et REST.

Rappelons quelques principes des interfaces REST :

- Le client accède à des ressources. Les réponses aux requêtes sont des représentations des ressources.
- Le client doit gérer l'état.
- Les requêtes indiquent des actions sur ces ressources. En utilisant les verbes du protocole http :
    - Créer (create) => POST
    - Afficher (read) => GET
    - Mettre à jour (update) => PUT

- Supprimer (delete) => DELETE

Créons maintenant notre propre service REST. Pour cela nous utiliserons l'outil « JSON Server » écrit en Node.js et disponible via npm. JSON Server est un outil qui permet d'exposer et manipuler un document JSON via une interface REST.

Regardez la documentation de JSON Server :
https://www.npmjs.com/package/json-server

Utilisez JSON-server pour créer, à partir d'un document JSON, un service permettant la gestion d'un portfolio ! C'est-à-dire qui permet :

- De gérer les produits dans le portefolio
- D'ajouter des produits dans le porte folio
- De retirer des produits dans le portefolie

Nous considérerons que vous gérez une librairie !

Vous noterez ici, que sur la seule base de la structure des ressources que l'on souhaite exposer, nous pouvons générer une API REST. Comme l'outil sait que (i) nous souhaitons construire une API de type REST permettant de faire des opération CRUD sur des données et (ii) que ces données sont au format JSON. Il est possible de créer l'interface REST automatiquement ! Dans ce type de services, le couplage entre les ressources et l'API est parfois fort et il est important de bien penser la structure des données (ressources).

Validez avec moi votre structure de données !

Tester votre service à l'aide de « cURL » (Si besoin, vous pouvez le télécharger sur http://curl.haxx.se/download.html).

Voici quelques exemples d'utilisation de cURL :

Pour déposer un document, envoyer un PUT au « EndPoint » identifié par 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 POST -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 PUT -d '{"name": "Iced Mocha", "size": "Grandé", "tasty":
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'
```

Testez votre service portefolio :

- Ajoutez un livre.
- Supprimez un livre.
- Chercher les informations à propos d'un livre à partir de son identifiant.
- Obtenir la liste de tous les livres.
- Listez tous les livres dans la catégorie Fiction

Pour aller plus loin : votre premier service SOAP !

Cette partie du TP se fera sous la forme d'une démo live.

Les étapes pour reproduire la démo sont les suivantes :

1. Dans Visual Studio, créez un « WCF Service Library project » (Bibliothèque de Service WCF). Si vous ne trouvez pas ce type de projet dans vos template Visual Studio, cela indique probablement que la Windows Communication Foundation n'est pas installée. Si c'est le cas, allez à la fin de la liste des templates et cliquez sur « Add more tools and features ». Une nouvelle fenêtre devrait s'ouvrir. Dans celle-ci, cliquez sur l'onglet « Individual component », sélectionnez « Windows Communication Foundation » et cliquez sur « Modify ».
2. Une fois le projet créé, vous pouvez déjà le démarrer, il contient un exemple complet de service SOAP. Vous venez de lancer votre premier service SOAP !
3. Lorsque vous lancez votre service un client de test est lancé automatiquement. Vous pouvez l'utiliser pour faire des appels aux méthodes GetData(). Et GetDataUsingContract()

L'exemple généré par Visual Studio contient trois fichiers :

- **IService1.cs :** Ce fichier contient l'interface de votre service. Il servira entre autres à générer le WSDL de votre service.
- **Service1.cs :** Ce fichier contient l'implémentation du service et donc les méthodes qui seront exécutées lors de la réception de requêtes SOAP.
- **App.config :** Il s'agit d'un fichier XML qui décrit les endpoints du service. Il contient entre autres une « base address », qui correspond à l'url à laquelle votre service écoute.

Vous pouvez visualiser le contrat de votre service en accédant à l'adresse : `baseAddress?WSDL`.

Votre service

Nous allons maintenant créer notre propre service permettant de faire des opérations mathématiques toutes simple.

1. Créez un nouveau projet pour votre service, vous l'appellerez MathLibrary
2. Renommez IService en IMathsOperations. L'interface de notre service pourra être la suivante :

```
namespace MathLibrary
{ public interface IMathsOperations
  {
     int Add(int num1, int num2);
     int Multiply(int num1, int num2);
  }
}
```

3. Renommez le fichier Service en MathOperations
4. Pour que IMathsOperations devienne une interface de service SOAP, ajouter un attribut **[ServiceContract]**.
5. Aussi toute opération que vous souhaitez rendre visible pour le client doit être décorée avec l'attribut **[OperationContract]. [ServiceContract]** et **[OperationContract]** sont inclus dans l'espace de nom System.ServiceModel.
6. Implémentez le service dans MathsOperations.cs
7. Testez votre service avec le client de Visual Studio
8. Observez le WSDL de votre service depuis votre navigateur.

Finalement ce que nous avons fait dans cet exercice est très proche de ce que nous avons réalisé avec json-server dans l'exercice précédent. Ici nous avons créé notre service depuis notre code : l'interface est créée à partir du code et de quelques annotations. Dans le cas du service REST, l'interface était créée à partir des « ressources ». Cela illustre la différence de point de vue entre interfaces de type REST vs RPC. Bien entendu il y a d'autres différences ! (cf. cours)