Mise en place d’un service web Java avec une API SOAP

Dans ce rapport synthétique, je vais décrire les manipulations à réaliser pour mettre en place un service web local en Java, comment tester des méthodes simples et créer un objet Java via l’API.

Les outils utilisés sont :

- IntelliJ IDEA (pour le développement du projet)
- Java 8 (nommé JDK 1.8 Corretto sur IntelliJ IDEA)
- SOAP UI (pour tester l’API SOAP et les méthodes du service web)

PARTIE 1 : Mise en place du service web et compréhension des notions

1.1 Création d’un nouveau projet

La première étape consiste à créer un nouveau projet Java, ici en version 8 qui possède les packages/méthodes présents dans ce TP. Cette version inclut notamment JAX-WS et JAXB, que l’on reverra plus tard, pour simplifier le développement de notre service web SOAP.

L’utilisation d’un IDE comme IntelliJ IDEA permet de créer simplement et efficacement une structure de projet propre, avec notamment des options de test et autre programmes.

1.2 Création des classes Application et MonServiceWeb

Dans le dossier « src », on ajoute une classe nommée **Application**.
Cette classe sera notre classe principale contenant une méthode **main()**, qui sera utilisée pour démarrer l’application Java. C’est également dans cette classe que nous déposerons plus tard l’Endpoint, c’est-à-dire l’objet chargé d’exposer notre service web sur une URL locale.

Une seconde classe, **MonServiceWeb**, contient la logique métier du service. Il s’agit d’un service SOAP standard : cette classe sera annotée afin que Java puisse en générer automatiquement un WSDL (Web Services Description Language) ainsi qu’un schéma XSD (XML Schema Definition). C’est également dans cette classe que l’on va pouvoir écrire les différentes méthodes utilisable. Comme exemple on va réaliser une méthode **conversion()**, qui prendra un double en entrée et renverra un autre double.

1.3 Compréhension des technologies JAX-WS et JAXB

JAX-WS (Java API for XML Web Services)

C’est l’API Java permettant de **créer des services web SOAP**. Elle s’appuie sur des annotations destinées à : exposer une classe Java comme service web, définir ses méthodes comme opérations SOAP, et générer automatiquement le WSDL décrivant ce service.

JAXB (Java Architecture for XML Binding)

JAXB permet de **convertir des objets Java en XML** (sérialisation) et **de lire du XML en objets Java** (désérialisation). JAX-WS utilise JAXB en interne pour représenter les messages SOAP en Java.

En résumé : **JAX-WS =** création du service web, **JAXB =** transformation XML en objets Java.

1.4 Ajout de l'annotation @WebService

La classe MonServiceWeb doit être annotée avec **@WebService**. On peut également définir un **targetNamespace**, ce qui permet de personnaliser l'espace de nom du service dans le WSDL généré. Cela facilite la structuration et l'identification du service, notamment pour des projets composés de plusieurs services web. En tant que service web, les méthodes de la classe seront exposée comme des opérations SOAP grâce aux annotations JAX-WS par défaut.

1.5 Déploiement du service via Endpoint dans la classe Application

Dans la méthode **main()**, on ajoute un objet **Endpoint**. Ce composant permet de publier le service web **sans serveur d'application** (comme Tomcat ou Glassfish). L'Endpoint est configuré pour exposer le service sur une URL. Dans notre cas, on utilisera : http://localhost:8888/ . Endpoint permet donc de facilement tester notre application en locale durant la phase de développement.

On peut maintenant lancée l'application, ce qui va activé le service web localement.

1.6 Test du service web

Après lancement du programme, on peut tester le WSDL et le XSD ne notre application. Pour WSDL, il faut aller sur l'URL que l'on a indiqué dans notre main, et en y ajoutant : ?wsdl. Cette URL permet d'obtenir automatiquement le WSDL généré par JAX-WS.

WSDL décrit : le service, les ports, le type de données, les messages SOAP, ainsi que la structure globale du service.

On peut y vérifier que la méthode **conversion** apparaît, mais **le WSDL ne contient pas la logique interne de la méthode**, seulement son existence et les types utilisés.

On peut également vérifier le XSD en ajoutant cette fois-ci : ?xsd=1. Le fichier XSD généré décrit les **types de données** utilisés par le service. Il fournit la définition XML des objets échangés.

1.7 Test de la méthode conversion() via SOAP UI

Pour effectuer un test complet du service, on a besoin de tester ses opérations SOAP sur un client. Pour ce faire, on utilise le logiciel **SOAP UI**.

Pour commencer, il faut créer un nouveau projet SOAP, dans lequel on indique le nom ainsi que l'URL en y incluant bien ?wsdl à la fin. SOAP UI va ensuite généré automatiquement une requête SOAP pour chaque méthode contenu dans le WSDL. En ouvrant la requête, on peut y voir un

fichier XML dans lequel on peut y renseigner des valeurs à nos arguments. Une fois fait, tester la requête pour obtenir un autre fichier XML contenant le résultat de la requête.

Ce test valide que le service SOAP est fonctionnel et correctement exposé.

1.8 Rechargement de l'application et du client SOAP UI

Lorsque l'on effectue une modification, comme un ajout de méthode ou de nouveaux types, il faut relancer l'application Java pour republier le service avec les nouvelles informations, puis rafraîchir le « PortBinding » du projet client dans SOAP UI afin que lui aussi recharge le WSDL mis à jour.

1.9 Autres annotations JAX-WS et JAXB

Il est également possible d'annoter les méthodes et les paramètres des méthodes pour mieux correspondre à ce que le client attend, ou pour simplement clarifier son utilisation.
Pour ce faire, on utilise :

- **@WebMethod** : pour annoter explicitement une méthode exposée.
- **@WebParam** : pour définir, par exemple, le nom des paramètres dans le WSDL.

Il est aussi possible de créer des annotations permettant d'ajouter de nouveaux comportements ou de mieux configurer les paramètres exposés par SOAP.

PARTIE 2 : Manipulation d'un objet Java en XML via JAXB

Pour enrichir notre API, on va essayer d'envoyer/créer des objets/instances Java.

2.1 Création de la classe Etudiant

On va donc créer une classe **Etudiant** qui implémente la classe « Serializable ». Cela rend la classe sérialisable. On ajoute ensuite des variables simple : un **identifiant**, un **nom** et une **moyenne**. On implémente également un constructeur ainsi que les getters et setters appropriés.
Pour pouvoir convertir cette classe en XML, elle doit être précédé de l'annotation **@XMLRootElement** pour que JAXB puisse l'interpréter et l'utiliser comme racine d'un XML.

2.2 Ajout de la méthode getEtudiant dans MonServiceWeb

On va donc ajouter une nouvelle méthode **getEtudiant** qui va simplement créer et retourner une instance de **Etudiant** prérempli par (1, « Mario », 19) correspondant à (identifiant, nom, moyenne).

Il suffit maintenant de relancer notre application ainsi que SOAP UI pour effectuer nos tests sur cette nouvelle méthode et voir que cela renvoi bien un objet Etudiant avec les paramètres prérempli.

Conclusion

Ce TP permet donc de mieux comprendre le fonctionnement d'un service web SOAP en Java grâce à JAX-WS et JAXB, sans utiliser de serveur d'application. Grâce à @XMLRootElement, on peut également générer du code XML basé sur un objet Java. Ce modèle reste simple et très pratique pour le prototypage ou la formation aux services web basés sur XML/SOAP.