# Développement d'un Service Web SOAP en Java sur Linux

Riham Kaddour Bakir

22 novembre 2025

## Table des matières

# 1 Introduction

Ce TP a pour objectif de développer un service Web SOAP en Java, en utilisant JAX-WS et JAXB,( machine utlisée Linux ) . la version utlise de jdk est 8 (de Java Temurin ), ansi que IntelliJ IDEA comme éditeur de text . le développement du service, son déploiement et son test avec SoapUI.

# 2 Création d'un projet Java simple

Dans cette première étape, nous allons créer un programme Java très simple afin de vérifier que :

— le JDK 8 est bien configuré,
— IntelliJ compile et exécute correctement le projet,
— nous avons une base pour ajouter notre service SOAP plus tard.

```java
public class Application {
    public static void main(String[] args){
        System.out.println("Debut du deploiement du service Web...");
    }
}
```

aucune logique de service web n'est encore créée. Nous vérifions seulement que l'environnement fonctionne avant d'ajouter SOAP

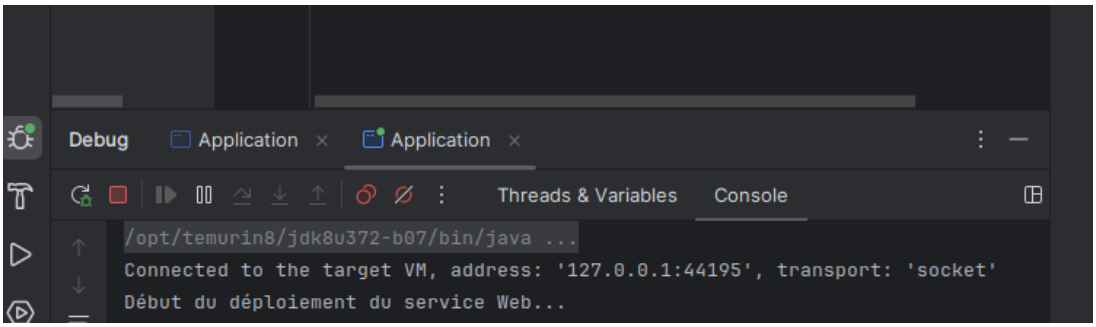


FIGURE 1 – affichage du console.

# 3 Développement du service Web SOAP avec JAX-WS

Dans cette étape, nous allons créer notre service SOAP grâce à JAX-WS. Cela consiste à :

— définir une classe Java,
— y ajouter des méthodes,
— les annoter pour les rendre accessibles en SOAP,
— indiquer l'espace de nommage (namespace) du service.

```java
package com.example.webservice;

import javax.jws.WebMethod;
import javax.jws.WebParam;
```

```java
import javax.jws.WebService;

@WebService(targetNamespace = "http://example.com/webservice")
public class MonserviceWeb {

    @WebMethod
    public double conversion(@WebParam(name = "MT") double montant)
        {
        return montant * 0.9;
    }

    @WebMethod
    public double somme(@WebParam(name = "A") double a, @WebParam(
       name = "B") double b) {
        return a + b;
    }
}
```

# 4 Déploiement du service SOAP

Maintenant nous allons publier notre service SOAP sur un serveur embarqué. JAX-WS permet de déployer un service SOAP sans Tomcat, juste avec Java SE.

```java
import javax.xml.ws.Endpoint;
import com.example.webservice.MonserviceWeb;

public class Application {
    public static void main(String[] args) {
        System.out.println("Debut du deploiement du service Web..."
           );
        String url = "http://localhost:8888/";
        Endpoint.publish(url, new MonserviceWeb());
        System.out.println("Le service web est deploye");
    }
}
```

```
/opt/temurin8/jdk8u372-b07/bin/java ...
Connected to the target VM, address: '127.0.0.1:44195', transport: 'socket'
Début du déploiement du service Web...
Le service web est déployé
```

FIGURE 2 – affichage du console .

## Publication du service SOAP

La fonction `Endpoint.publish()` permet de :
— démarrer un mini-serveur HTTP interne à Java ;
— publier le service SOAP développé ;

— rendre automatiquement disponible le fichier WSDL associé.

### URL d'accès au service

Une fois l'application lancée, le service est accessible via l'URL suivante :

`http://localhost:8888/?wsdl`

Cette adresse permet d'afficher le fichier *WSDL* généré automatiquement par JAX-WS, décrivant la structure complète du service Web. contient plusieurs sections importantes :

— **Les messages** : ils décrivent les paramètres et les retours des opérations du service, par exemple `conversion`, `somme`, etc.
— **Le portType** : il regroupe les opérations disponibles dans le service Web. Ici, `MonserviceWeb` contient les opérations `conversion` et `somme`.
— **Les opérations** : chaque opération correspond à une méthode annotée `@WebMethod` dans la classe Java.
— **Le targetNamespace** : il identifie l'espace de nom unique du service, ici `http://example.com/webservice`.

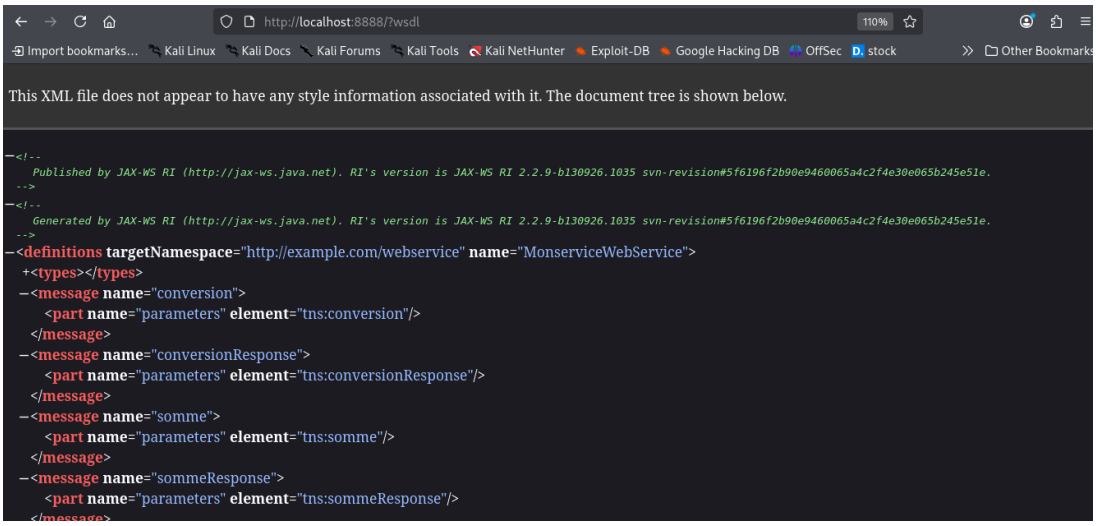


FIGURE 3 – affichage du lien .

Ainsi, l'affichage du WSDL confirme que le service SOAP est correctement déployé et que JAX-WS a généré la description complète du service nécessaire aux clients SOAP (comme SoapUI).

# 5 Test du service SOAP avec SoapUI

## 5.1 Installation via Flatpak

```
sudo apt install flatpak
sudo flatpak install flathub org.soapui.SoapUI
flatpak run org.soapui.SoapUI
```

## 5.2 Installation via l'installateur officiel

```
chmod +x SoapUI-x64-<version>.sh
./SoapUI-x64-<version>.sh
./soapui.sh
```

## 5.3 Création d'un projet SOAP dans SoapUI

Pour tester le service Web que nous avons développé, nous utilisons l'outil **SoapUI**, qui permet d'importer un service SOAP à partir de son fichier WSDL et d'invoquer ses différentes opérations.

### 1. Création d'un nouveau projet SOAP

Après avoir lancé SoapUI, nous créons un nouveau projet via l'option *File > New SOAP Project*. Une fenêtre demande alors l'URL du WSDL à importer. Nous indiquons :

`http://localhost:8888/?wsdl`

SoapUI analyse alors automatiquement le document WSDL généré par JAX-WS et crée toutes les structures nécessaires : opérations, requêtes SOAP, schémas XML, etc.

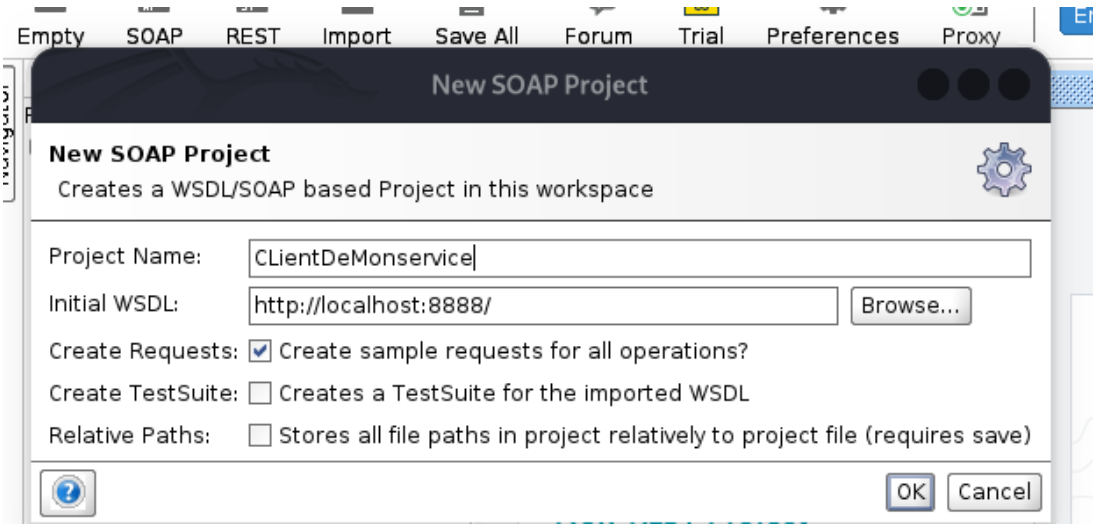


FIGURE 4 – Création d'un projet SOAP dans SoapUI à partir du WSDL local.

### 2. Test des méthodes du service

Une fois le projet généré, SoapUI affiche les différentes opérations disponibles dans le service Web. Dans notre cas, deux méthodes apparaissent :

— **conversion(mt)** : reçoit un montant et renvoie sa conversion (taux 0.9),

— **somme(a, b)** : renvoie la somme de deux valeurs numériques.

Pour tester une opération, il suffit de sélectionner une requête SOAP générée, puis de modifier les valeurs des paramètres dans le corps XML. En cliquant sur le bouton *Run*, SoapUI envoie la requête à notre service local et affiche la réponse SOAP retournée par JAX-WS.

La figure suivante illustre un exemple de test où SoapUI envoie une requête à l'opération `somme` et obtient correctement le résultat renvoyé par notre service :

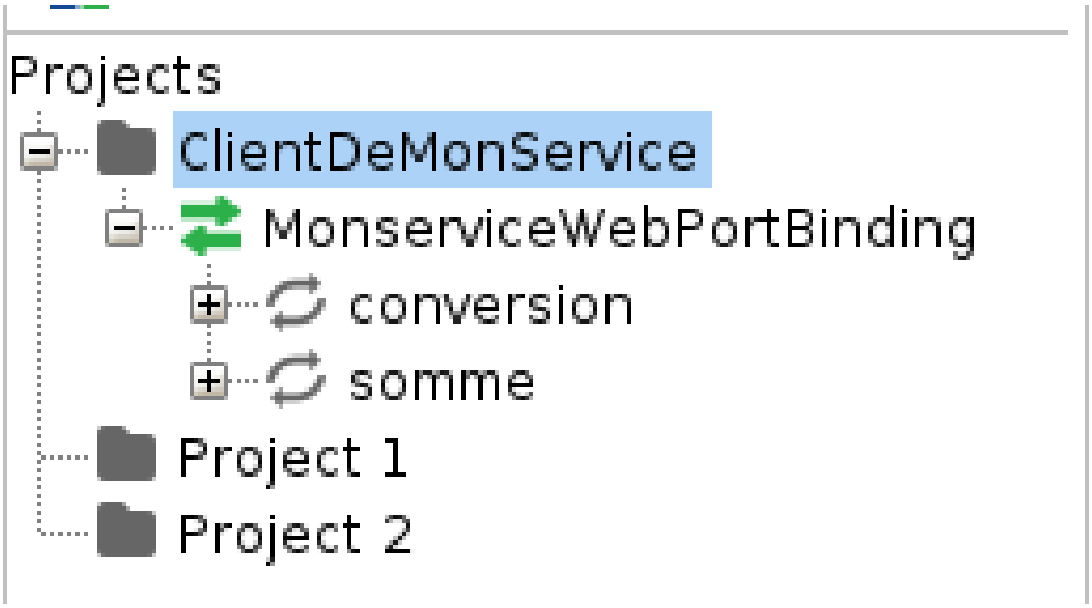


FIGURE 5 – Exécution d'une opération SOAP et affichage de la réponse du service.

Grâce à ces tests, nous pouvons valider le bon fonctionnement de notre service SOAP et vérifier que les valeurs renvoyées correspondent aux résultats attendus.

# 6 Manipulation d'objets avec SOAP et JAXB

Dans la deuxième partie de ce travail, l'objectif est d'étendre les fonctionnalités du service Web afin de ne plus manipuler uniquement de simples types numériques, mais de pouvoir échanger des objets complets entre le client et le serveur. Pour cela, nous allons introduire la notion de **JavaBean**, ainsi que les annotations de **JAXB** permettant la conversion automatique entre objets Java et représentations XML dans les messages SOAP.

## 6.1 Définition d'un JavaBean : la classe `Etudiant`

Pour illustrer la manipulation d'objets via SOAP, nous définissons une classe représentant un étudiant. Cette classe doit respecter les conventions d'un **JavaBean** :

— implémenter l'interface `Serializable`,
— posséder un constructeur sans paramètres,
— avoir des attributs privés,
— fournir des getters et setters publics.

De plus, pour que cette classe puisse être convertie automatiquement en XML grâce à **JAXB**, elle doit être annotée avec l'annotation `@XmlRootElement`. Cette annotation permet à JAX-WS de générer la structure XML correspondant à cet objet dans le WSDL.

Voici l'implémentation complète de la classe `Etudiant` :

```java
import javax.xml.bind.annotation.XmlRootElement;
import java.io.Serializable;

@XmlRootElement
public class Etudiant implements Serializable {
    private int identifiant;
    private String nom;
    private double moyenne;

    public Etudiant() {}

    public Etudiant(int identifiant, String nom, double moyenne) {
        this.identifiant = identifiant;
        this.nom = nom;
        this.moyenne = moyenne;
    }

    public int getIdentifiant() { return identifiant; }
    public void setIdentifiant(int identifiant) { this.identifiant
       = identifiant; }

    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }

    public double getMoyenne() { return moyenne; }
    public void setMoyenne(double moyenne) { this.moyenne = moyenne
       ; }
}
```

## 6.2 Intégration de l'objet dans le service Web

Nous enrichissons maintenant notre service SOAP en ajoutant une méthode permettant de renvoyer un objet de type `Etudiant`. Cette méthode sera ensuite visible dans le WSDL, et le client (SoapUI) pourra l'invoquer comme une opération SOAP classique.

```java
@WebMethod
public Etudiant getEtudiant() {
    return new Etudiant(1, "Karim", 15.7);
}
```

Après ajout de cette méthode, un nouveau type complexe XML est automatiquement ajouté dans le WSDL, correspondant à la structure de l'objet `Etudiant`.

```
  </message>
-<message name="getEtudiant">
    <part name="parameters" element="tns:getEtudiant"/>
  </message>
-<message name="getEtudiantResponse">
    <part name="parameters" element="tns:getEtudiantResponse"/>
  </message>
-<portType name="MonserviceWeb">
  -<operation name="convertir">
      <input wsam:Action="http://example.com/webservice/MonserviceWeb/
      convertirRequest" message="tns:convertir"/>
```

FIGURE 6 – Le WSDL met à jour la structure XML pour la classe Etudiant.

## 6.3 Appel de la méthode depuis SoapUI

Une fois le service redéployé, nous ouvrons SoapUI, qui détecte automatiquement la nouvelle opération `getEtudiant` grâce à la mise à jour du WSDL.

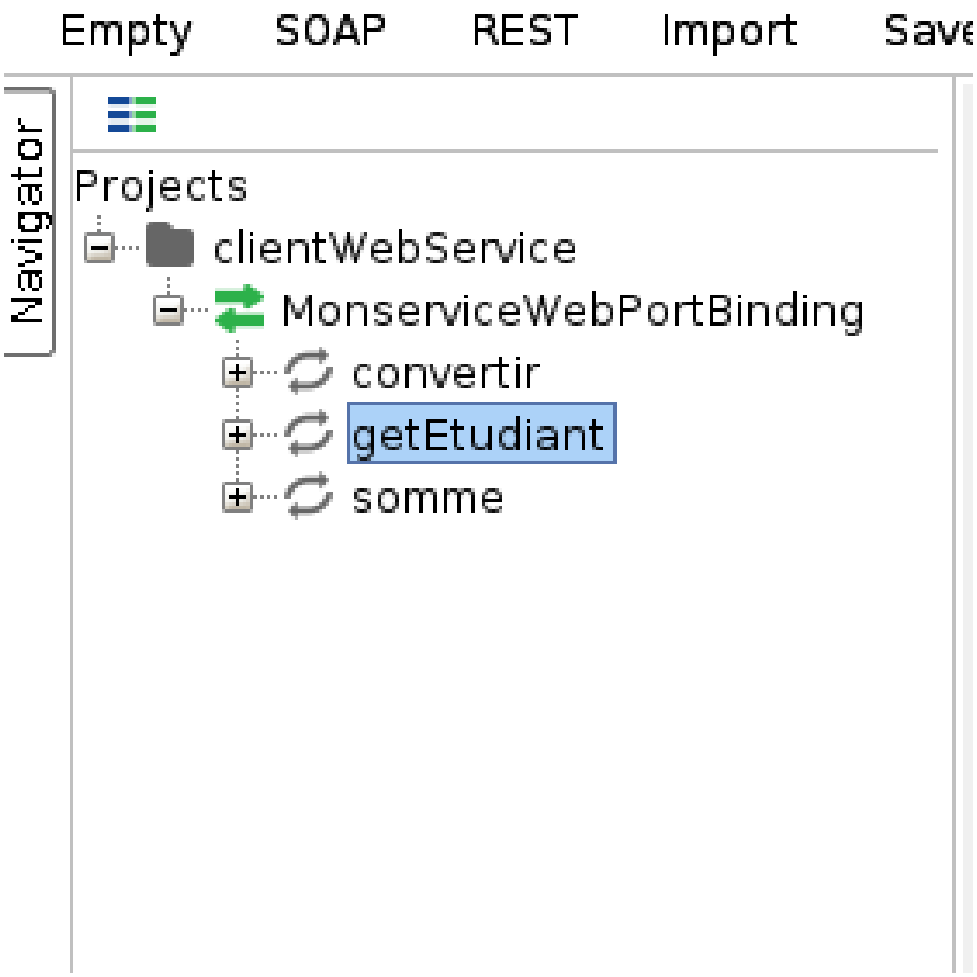


FIGURE 7 – La méthode `getEtudiant` apparaît dans SoapUI.

Nous pouvons alors exécuter la requête SOAP associée. Le serveur renvoie un objet sérialisé en XML selon la structure définie par JAXB.

Exemple de réponse obtenue :

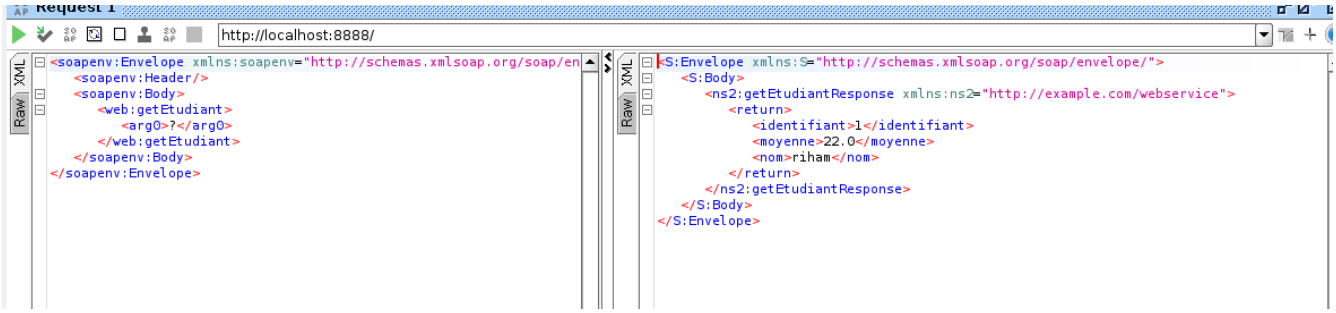


FIGURE 8 – Réponse SOAP contenant l'objet `Etudiant`.

Comme on peut l'observer, l'étudiant transmis par le serveur apparaît sous forme XML
.

# 7 Conclusion

Ce TP a permis de mettre en place un service Web SOAP complet en Java 8 sur Linux. Le service est déployé localement et testé via SoapUI. pour faire la création et le déploiement d'un service SOAP ainsi que l'utilisation d'outils de test.