Compt Rendu

5ème année
Ingénieur Informatique & Réseaux

TP6

SOAP

Réalisé par : *IMANE Chakrellah*
YASSINE Ech-chaoui

Prof : *Pr.Oussama*
Classe: *5IIR-11*

2025-2026

Table des matières

Tout les TPs Réalisés ce trouve dans le lien suivant qui contient le projet + CR :

https://github.com/CHAKRELLAH44/JEE-ARCHITECTURE.git

Objectif du TP

L'objectif de ce TP est de comprendre et d'implémenter un service Web utilisant le protocole SOAP et l'API JAX-WS.
Plus précisément, ce TP permet de :

- Comprendre l'architecture SOAP et la structure des messages SOAP.
- Comprendre le rôle et la structure d'un fichier WSDL.
- Développer un service web SOAP (serveur) en Java avec JAX-WS.
- Tester les opérations via SOAP UI.
- Générer un client SOAP automatiquement à partir du WSDL.
- Tester le service web depuis un client Java (JUnit + Stub généré).

Ce TP introduit la logique des services web "orientés contrat" et l'échange de données via XML.

1. Architecture du projet

1.1 Structure du projet

Le projet est organisé en différentes couches :

- `service.model` : contient le modèle métier (classe Article).
- `service` : contient l'interface métier et son implémentation (IService / ServiceImpl).
- `presentation` : contient le contrôleur SOAP exposé comme service web.
- `Main` : point d'entrée de l'application qui publie le service SOAP.

1.2 Description des composants

- **Article** : objet métier représentant un article (id, description, prix, quantité).
- **IService** : interface définissant les opérations du service (CRUD).
- **ServiceImpl** : implémentation contenant une base de données simulée en mémoire.
- **ArticleSoapController** : classe annotée avec `@WebService` exposant les méthodes SOAP.
- **Main** : démarre un serveur JAX-WS sur une URL fixe et publie le service web.

1.3 Rôle de chaque couche / module

Couche	Rôle
Model	Définition des objets métier échangés en XML
Service	Logique métier : gestion des articles
Présentation (SOAP controller)	Exposition des méthodes en tant que Web Services
Main	Publication du service sur un endpoint HTTP
Client	Consommation du service via un Stub généré automatiquement

2. Notions importantes

2.1 Concepts théoriques

- **SOAP** : protocole basé sur XML permettant l'échange structuré de données.
- **WSDL** : fichier XML décrivant le contrat du service (opérations, messages, types, protocole).
- **JAX-WS** : API Java permettant de créer et consommer des services SOAP.
- **Stub** : code client généré automatiquement à partir du WSDL pour appeler un service sans écrire XML manuellement.
- **Document/Literal** : style SOAP moderne utilisé par JAX-WS.

2.2 Technologies et outils utilisés

- Java 17
- JAX-WS RI (implémentation de référence)
- Lombok
- IntelliJ IDEA
- SOAP UI (tests d'un service SOAP)
- Maven
- JUnit 5 pour les tests client

2.3 Principes ou notions clés

- **WebService** : annotation transformant une classe en service SOAP.
- **Endpoint.publish()** : démarre un petit serveur HTTP embarqué pour exposer le service.
- **Proxy / Stub client** : permet à une application Java d'appeler le service comme une méthode locale.
- **WSDL-first vs Code-first** : ici, nous faisons du *Code First* → le WSDL est généré automatiquement.

3. Déroulement du TP

3.1 Étapes réalisées

1. Création du projet Maven et ajout des dépendances (JAX-WS, Lombok).
2. Création du modèle Article et de la couche service.
3. Mise en place du WebService `ArticleSoapController` avec les annotations JAX-WS.
4. Définition des 4 opérations SOAP :
   - getAll
   - getById
   - saveArticle
   - deleteById
   -
5. Publication du WebService via `Endpoint.publish()`.
6. Vérification du WSDL sur l'URL : **http://localhost:9999/ecommerce?wsdl**
7. Tests via SOAP UI.
8. Génération d'un client SOAP :

- via IntelliJ Ultimate (plugin Jakarta EE)
- ou via Maven (jaxws-maven-plugin)

9. Écriture des tests JUnit du client pour vérifier toutes les opérations.

4. Code source (extraits essentiels)

4.3 Workflow global

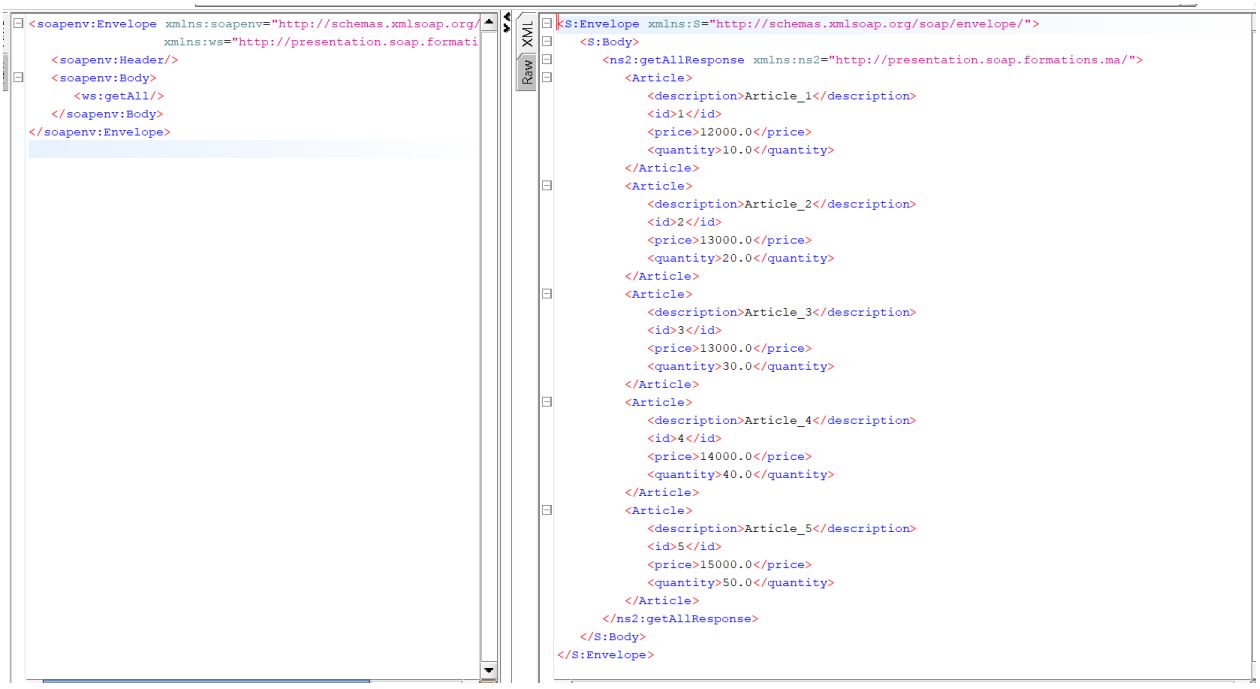
- L'utilisateur appelle le service via SOAP UI → XML envoyé au serveur.
- Le serveur JAX-WS reçoit la requête → exécute la méthode Java correspondante.
- Le résultat Java est transformé en XML → renvoyé sous forme de message SOAP.
- Le client SOAP (Stub) peut consommer ce résultat comme un simple objet Java.

5. Tests avec les endpoints5.1 Test SOAP – via SOAP UI

1) getAllArticles

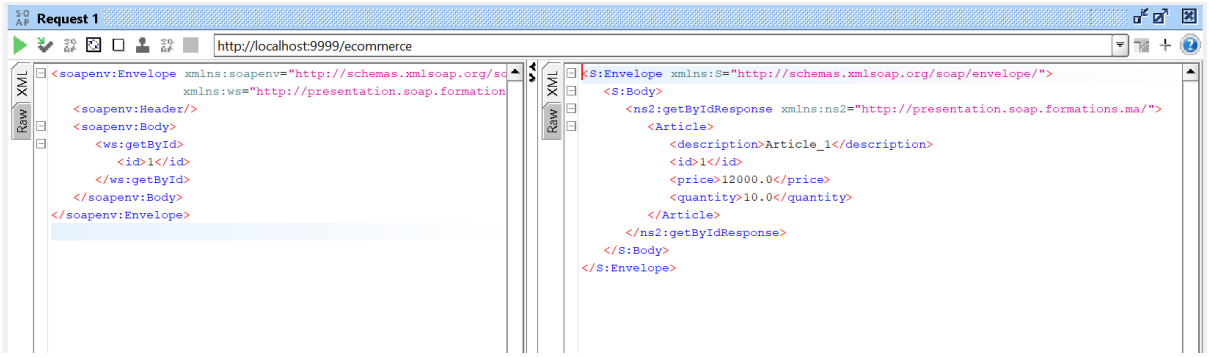
URL : `http://localhost:9999/ecommerce?wsdl`

→ Vérifie que la liste d'articles est retournée en XML.



2) getById

→ Envoie un XML contenant l'id → reçoit un Article complet.



3) saveArticle

→ Envoie un article en XML → vérifie qu'il est ajouté dans la liste.

4) deleteById

→ Supprime un article selon son ID → attend une confirmation textuelle.

Tests JUnit côté client

Chaque test appelle le Stub généré :

- **testGetAll()** → vérifie la taille de la liste.
- **testGetById()** → vérifie chaque champ de l'article retourné.
- **testDeleteById()** → attend le message exact.
- **testSave()** → ajoute un article et vérifie son existence.

7. Outils de visualisation

- **WSDL** : permet de visualiser la structure du service web
- **SOAP UI** : test des messages SOAP
- **Stub généré** : permet au client de visualiser le contrat sous forme de méthodes Java

8. Conclusion

Ce TP m'a permis de comprendre en profondeur le fonctionnement des services web SOAP, un standard encore largement utilisé dans les environnements bancaires, industriels et d'entreprise.

J'ai appris :

- Comment structurer un service web suivant une architecture en couches.
- Comment exposer un service SOAP avec JAX-WS.
- Comment fonctionne le WSDL et pourquoi il représente le contrat du service.
- Comment consommer un service en générant un client automatiquement.
- Comment tester un service SOAP avec SOAP UI et JUnit.

Limites du TP

- Le stockage est en mémoire (pas de base de données réelle).
- Le serveur SOAP est simple et non sécurisé (pas d'authentification).
- JAX-WS est une technologie ancienne, remplacée aujourd'hui par REST dans la majorité des cas.