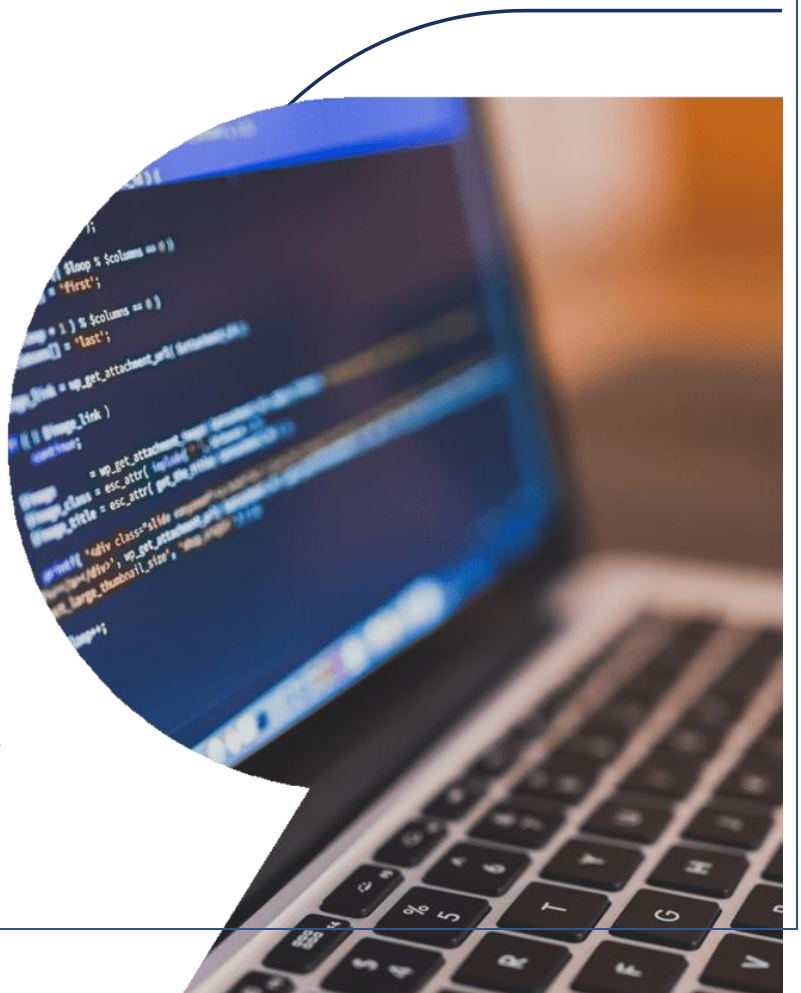
Rapport sur les Webservices : SOAP

RÉALISÉ PAR
Ikhlass BOUYAZID

Mr : Mohamed YOUSSFI



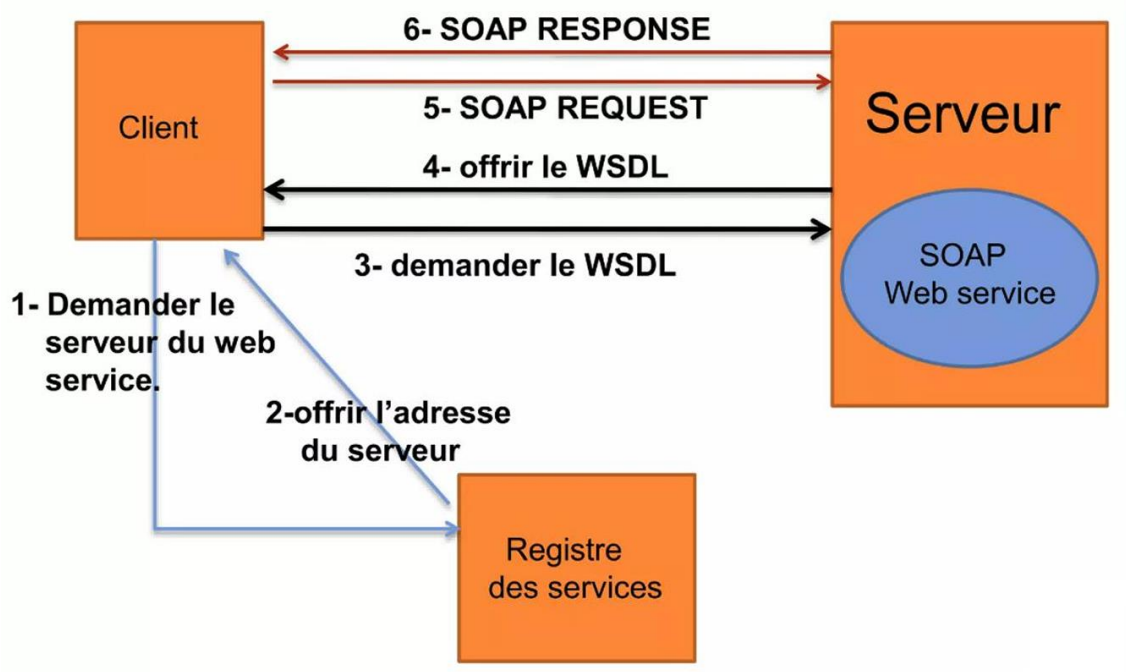
2023-2024

PLAN

Web Services

Un service web est un protocole d'interface informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone. Le protocole de communication est défini dans le cadre de la norme SOAP dans la signature du service exposé (WSDL). Actuellement, le protocole de transport est essentiellement TCP (via HTTP).

SOAP (Simple Object Access Protocol)



Cas pratique

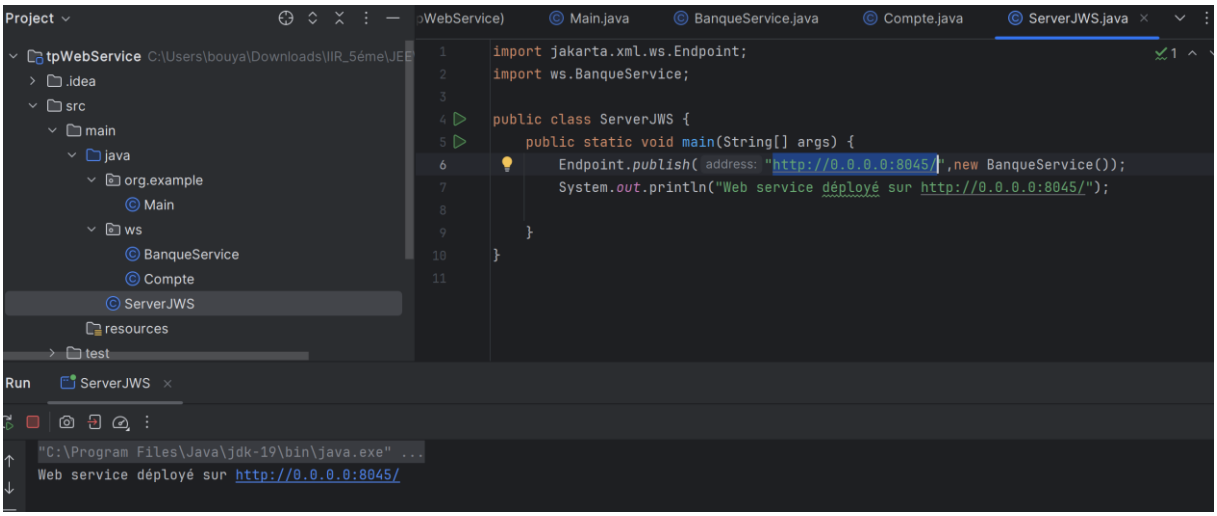
Pour commencer ,on crée une classe POJO, ou "Plain Old Java Object," :

```java
import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebService;

import java.util.Date;
import java.util.List;
@WebService(serviceName = "BanqueWS")

public class BanqueService {

//POJO Plain Old Java Object
    @WebMethod(operationName = "convert")
    public double conversion(@WebParam(name = "montant") double mt) {
        return mt * 10.54; }

    @WebMethod
    public Compte getCompte(@WebParam(name = "code ") int code) {
        return new Compte(code, solde: Math.random() * 9888, new Date());

    }

    no usages
```

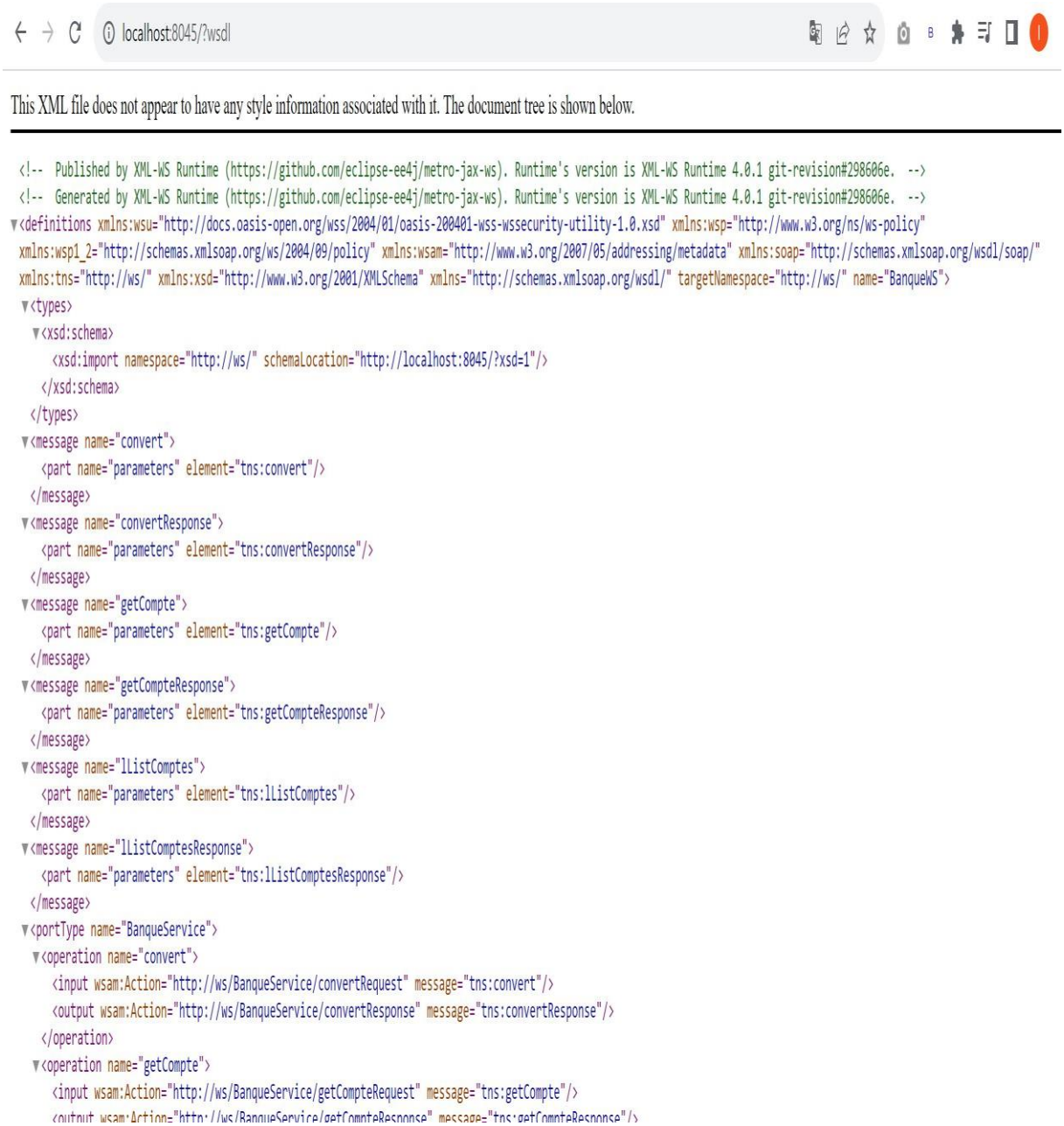
il nous faut donc d'abord ajouter une dépendance de JAX-WS qui va nous permettre d'ajouter les annotations adéquates à la classe.

```xml
<dependencies>
    <!-- https://mvnrepository.com/artifact/com.sun.xml.ws/jaxws-ri -->
    <dependency>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-ri</artifactId>
        <version>4.0.1</version>
        <type>pom</type>
    </dependency>

</dependencies>
```

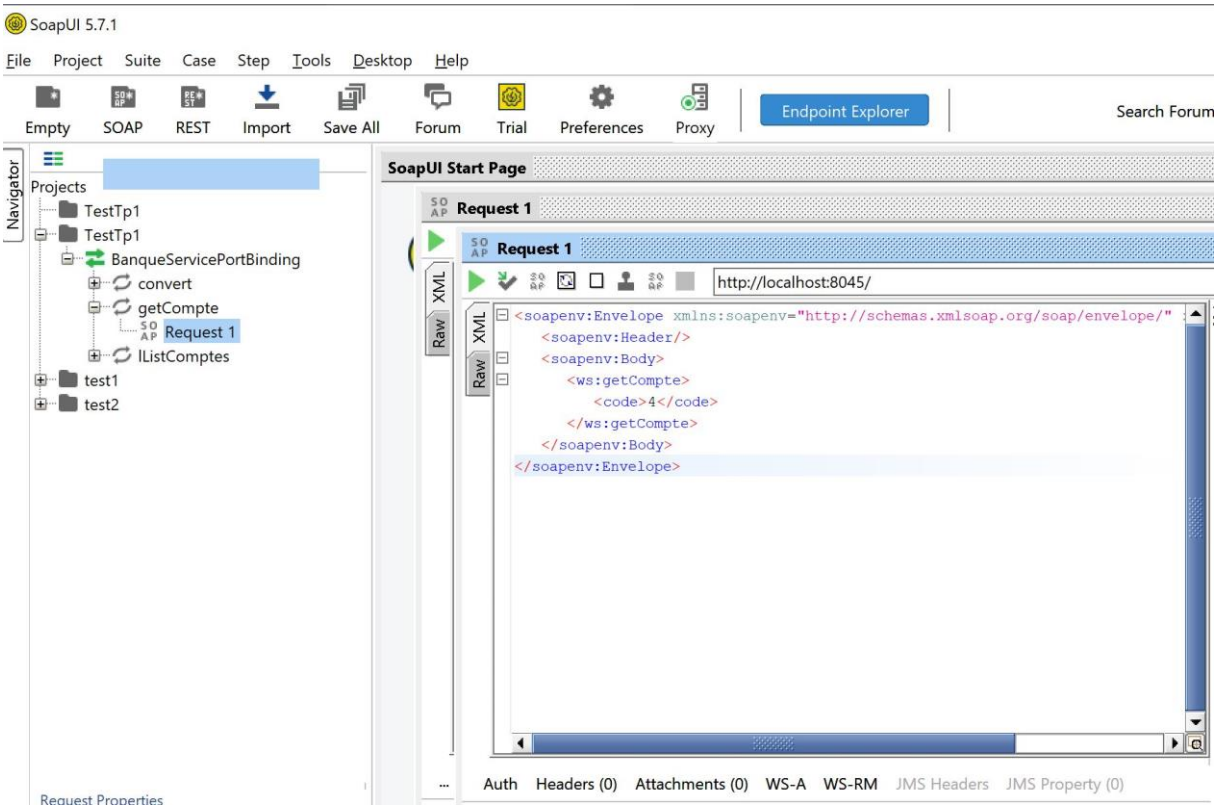
Pour configurer le serveur web pour prendre en charge le service web,on va créer la classe ServerJWS qui contient la méthode Endpoint.publish() ,

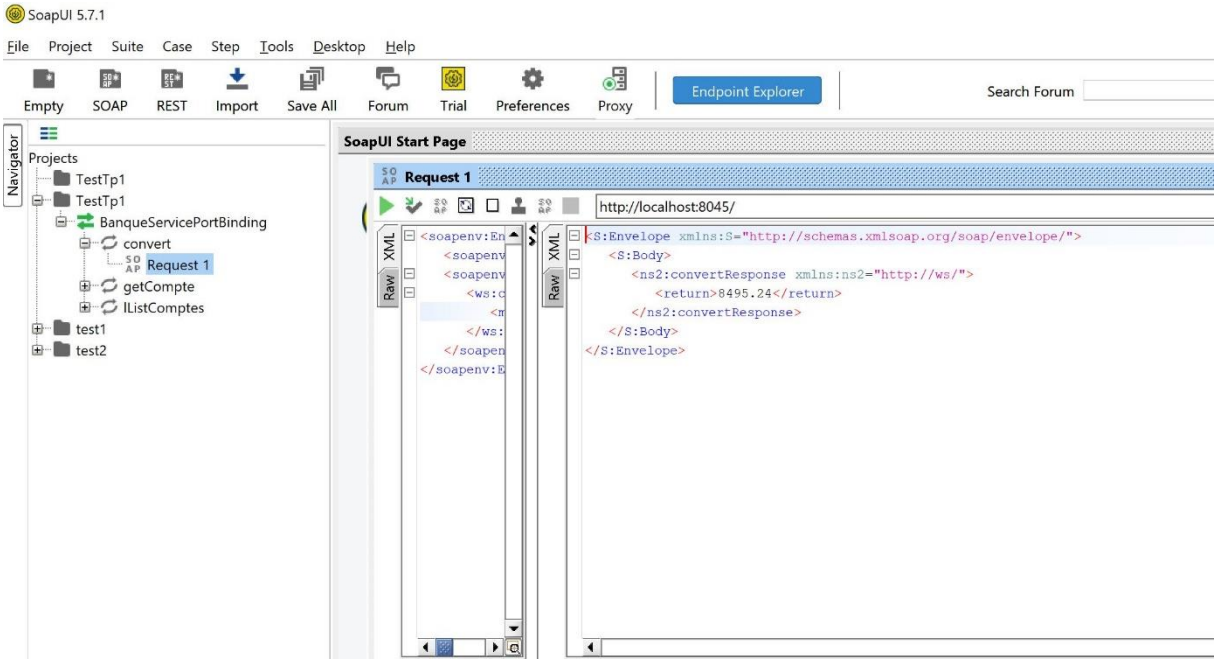


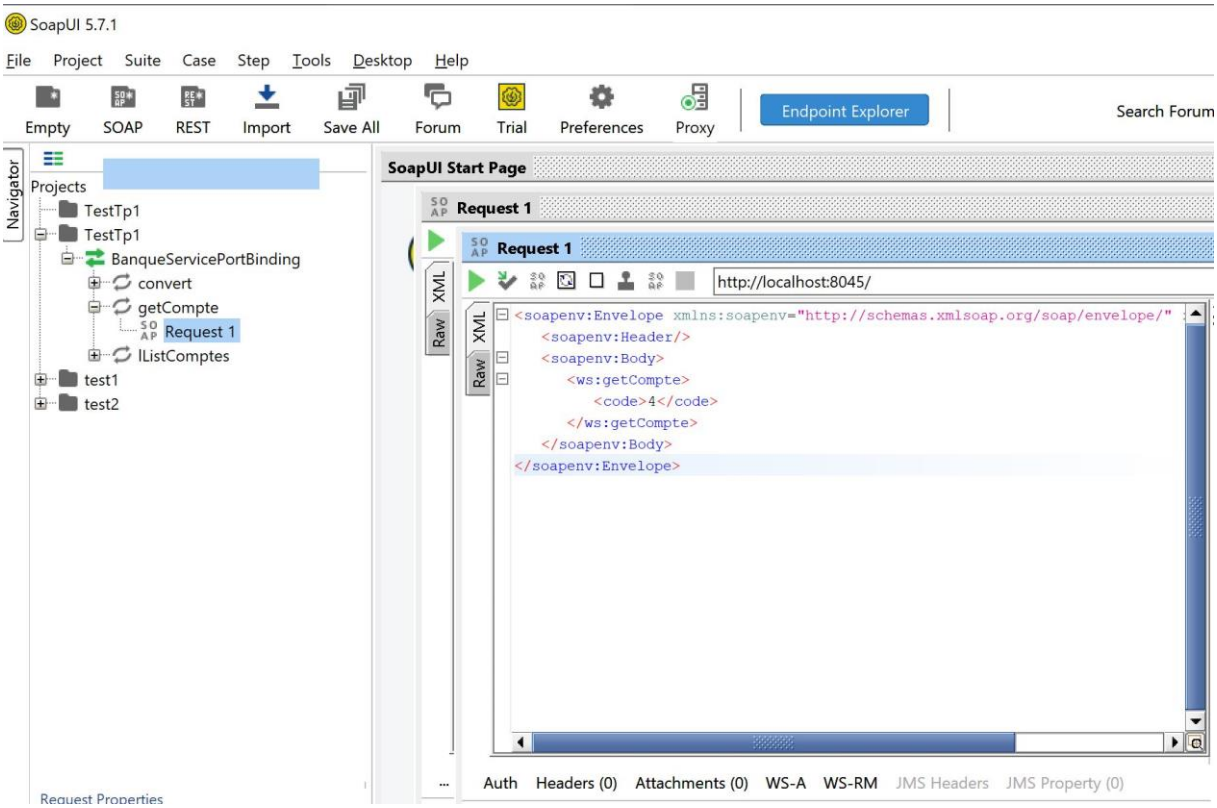
Le server est démarré on peut avoir donc notre WSDL :

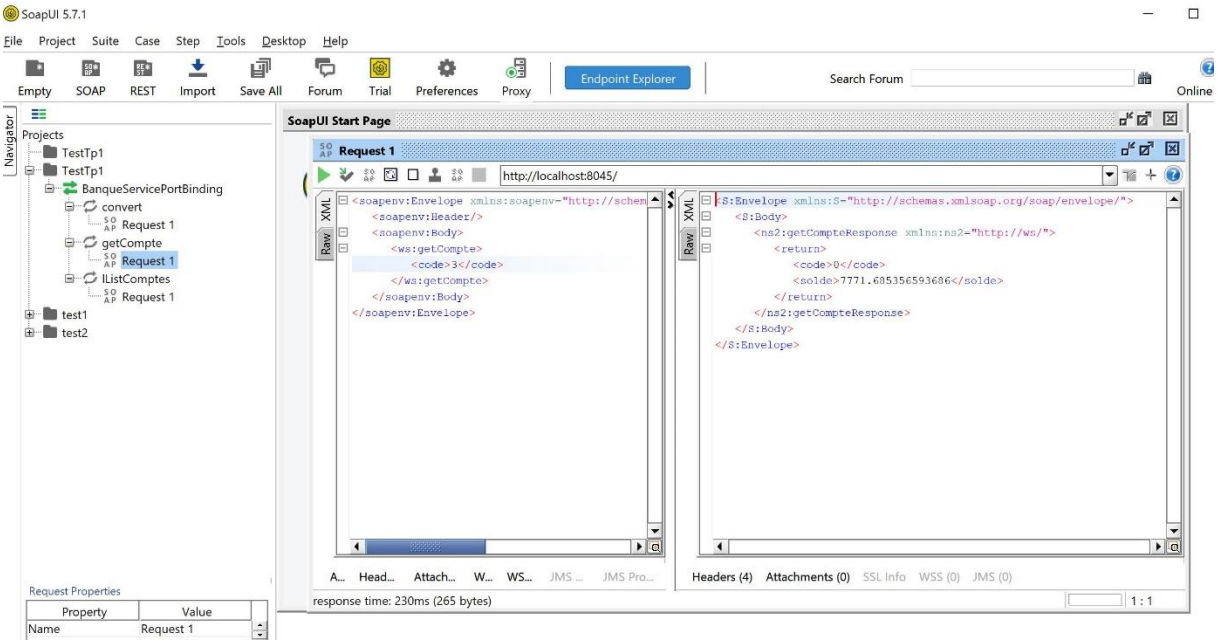


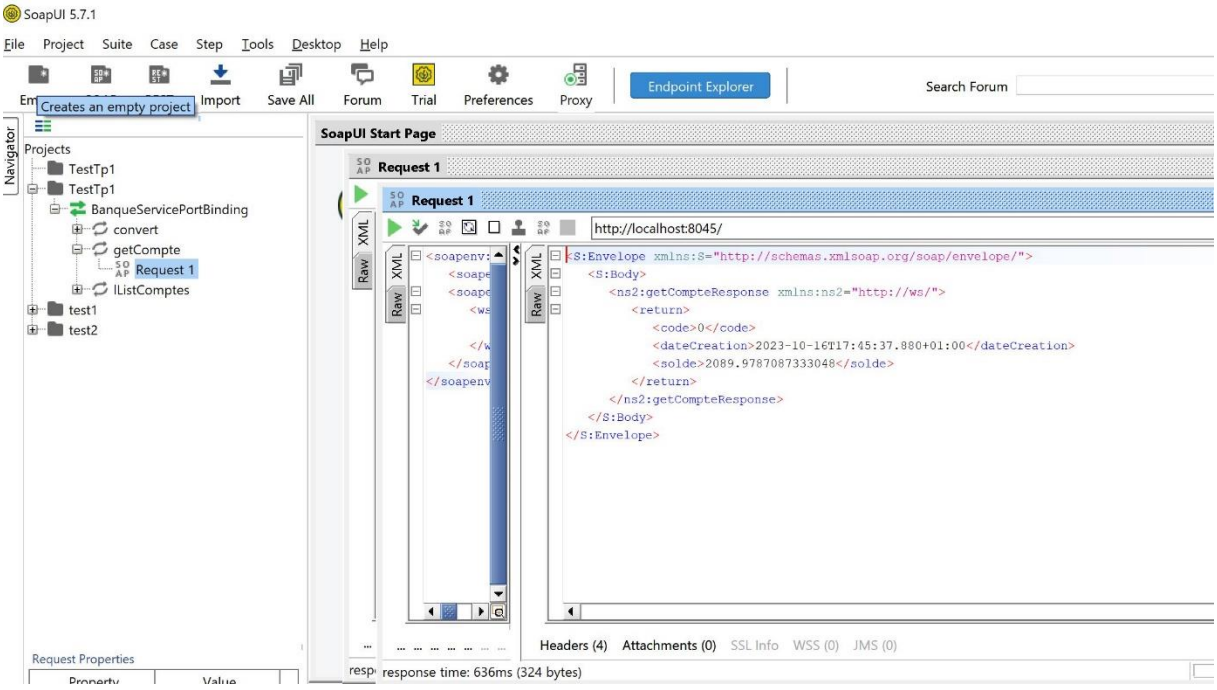
Pour tester notre serviceweb , on va utilisé SoapUI qui est largement utilisé pour tester, développer et simuler des services web SOAP (Simple Object Access Protocol) et REST (Representational State Transfer).






SoapUI 5.7.1
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Search Forum
Navigator
Projects
TestTp1
TestTp1
BanqueServicePortBinding
convert
getCompte
Request 1
IListComptes
test1
test2
Request Properties
SoapUI Start Page
Request 1
Request 1
http://localhost:8045/
XML
Raw
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
<soapenv:Header/>
<soapenv:Body>
<ws:getCompte>
<code>4</code>
</ws:getCompte>
</soapenv:Body>
</soapenv:Envelope>
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0)


SoapUI 5.7.1
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Search Forum
Online
Navigator
Projects
TestTp1
TestTp1
BanqueServicePortBinding
convert
Request 1
getCompte
Request 1
IListComptes
Request 1
test1
test2
SoapUI Start Page
Request 1
http://localhost:8045/
<soapenv:Envelope xmlns:soapenv="http://schem
<soapenv:Header/>
<soapenv:Body>
<ws:getCompte>
<code>3</code>
</ws:getCompte>
</soapenv:Body>
</soapenv:Envelope>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:getCompteResponse xmlns:ns2="http://ws/">
<return>
<code>0</code>
<solde>7771.685356593606</solde>
</return>
</ns2:getCompteResponse>
</S:Body>
</S:Envelope>
Request Properties
Property
Value
Name
Request 1
Headers (4) Attachments (0) SSL Info WSS (0) JMS (0)
response time: 230ms (265 bytes)


SoapUI 5.7.1
File Project Suite Case Step Tools Desktop Help
Creates an empty project
Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Search Forum
Navigator
Projects
TestTp1
TestTp1
BanqueServicePortBinding
convert
getCompte
Request 1
IListComptes
test1
test2
SoapUI Start Page
Request 1
Request 1
http://localhost:8045/
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:getCompteResponse xmlns:ns2="http://ws/">
<return>
<code>0</code>
<dateCreation>2023-10-16T17:45:37.880+01:00</dateCreation>
<solde>2089.9787087333048</solde>
</return>
</ns2:getCompteResponse>
</S:Body>
</S:Envelope>
Request Properties
Headers (4) Attachments (0) SSL Info WSS (0) JMS (0)
response time: 636ms (324 bytes)

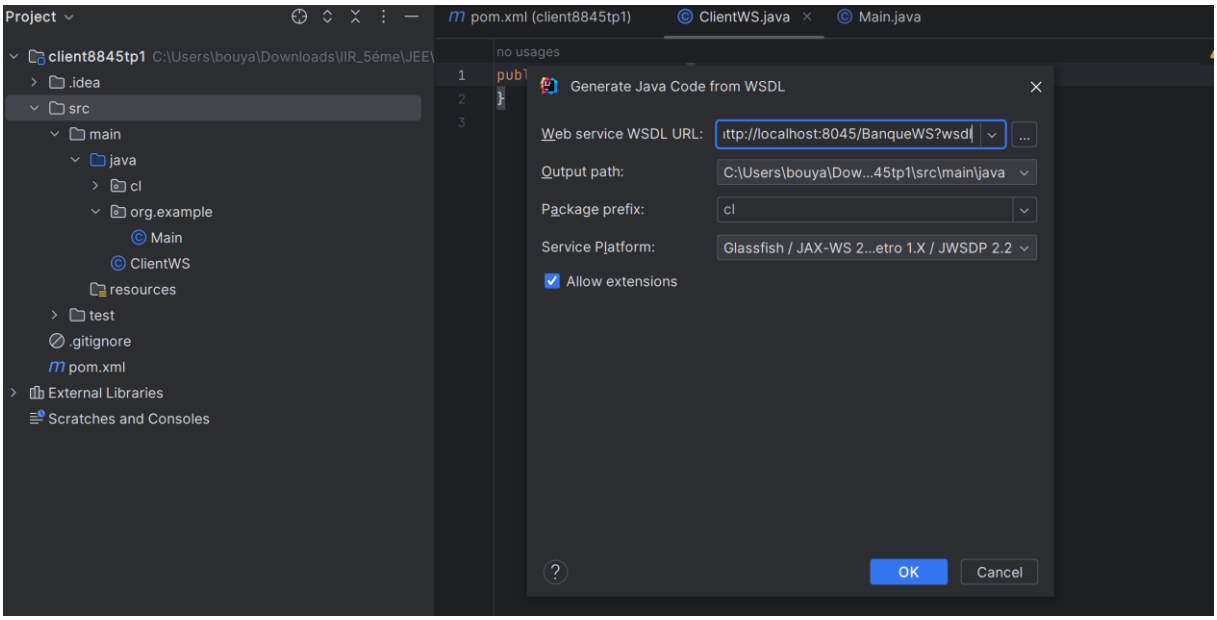
Pour créer un client Java et générer un proxy pour consommer un service web, nous allons utiliser JAX-WS (Java API for XML Web Services). Voici les étapes à suivre :

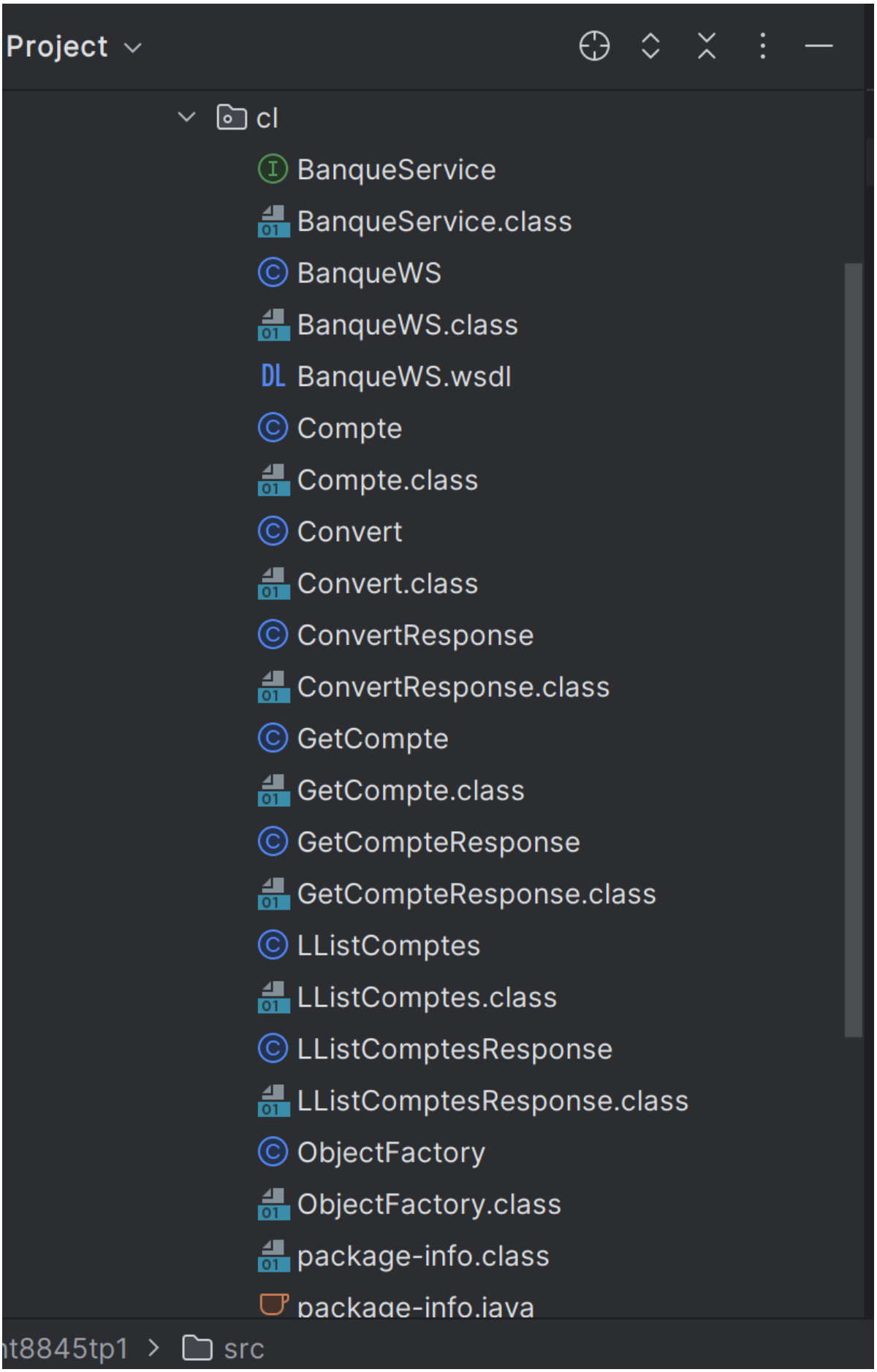
1. Créez un projet Java dans IntelliJ IDEA.

2. Générez le proxy à partir du WSDL

3. Créez une classe Java pour votre client web.

4. Créez un objet STUB qui représente un middleware pour consommer notre web service




Project
cl
BanqueService
BanqueService.class
BanqueWS
BanqueWS.class
BanqueWS.wsdl
Compte
Compte.class
Convert
Convert.class
ConvertResponse
ConvertResponse.class
GetCompte
GetCompte.class
GetCompteResponse
GetCompteResponse.class
LListComptes
LListComptes.class
LListComptesResponse
LListComptesResponse.class
ObjectFactory
ObjectFactory.class
package-info.class
package-info.java
t8845tp1
src

Conclusion

En Java, l'utilisation de JAX-WS (Java API for XML Web Services) est courante pour consommer des services web. Dans l'environnement de développement IntelliJ IDEA, nous générons un proxy à partir du fichier WSDL du service, ce processus engendre automatiquement la création de classes Java, y compris un Stub. Ce Stub agit comme une interface entre notre application cliente et le service distant, gérant la sérialisation et la désérialisation des données pour la communication.

Dans notre classe cliente Java, nous faisons usage de ce Stub pour invoquer les méthodes du service web distant. Le Stub établit la communication avec le Skeleton situé du côté du serveur. Le Skeleton est un composant résidant sur le serveur, servant d'intermédiaire entre le service web distant et l'objet distant réel qui implémente le service.

Lorsqu'une requête parvient au Skeleton, il la transmet à l'objet distant, lequel effectue le traitement requis. Une fois que l'objet distant génère une réponse, le Skeleton renvoie cette réponse au Stub, qui la réachemine vers le client. Le rôle du Skeleton est essentiel pour gérer la communication entre le client et le service distant du côté du serveur, permettant ainsi au client d'appeler les méthodes du service web de manière transparente