

Rapport sur les Webservices SOAP

ACHBAD Fatima

11/10/2023

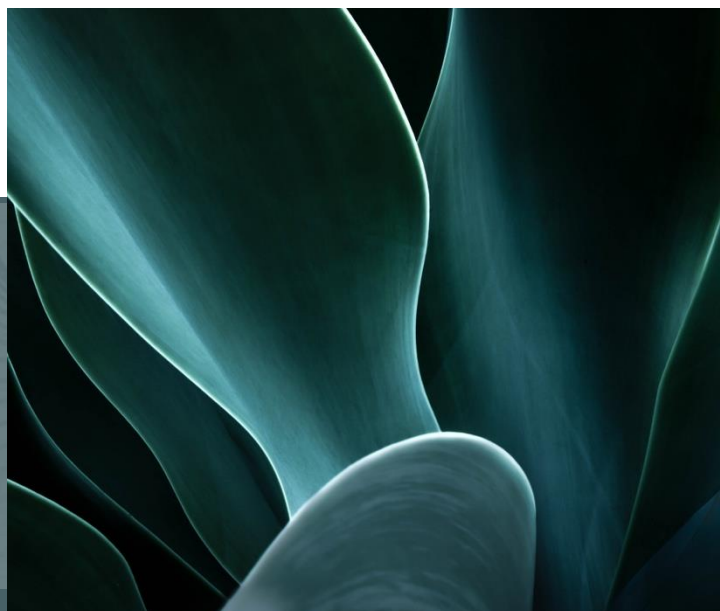
—

Mohamed YOUSSEFI

Introduction

Dans le monde d'aujourd'hui, les applications logicielles sont de plus en plus connectées entre elles. Cette interconnexion est rendue possible par les web services, qui permettent à des applications de communiquer et d'échanger des données entre elles, quel que soit leur langage de programmation ou leur plateforme.

SOAP (Simple Object Access Protocol) est un protocole de communication utilisé pour les web services. Il est basé sur XML et permet de transmettre des messages structurés entre des applications. SOAP est un protocole standard, soutenu par de nombreux langages de programmation et de Framework.





Objectifs du rapport

Le présent rapport vise à fournir une introduction aux web services SOAP , ainsi que des technologies à savoir : WSDL, UDDI et JAXWS permettent aux développeurs de créer et d'utiliser des services web qui peuvent être utilisés par des applications de n'importe quel langage de programmation ou de plateforme, en plus d'un cas d'utilisation pratique pour les utiliser ensemble.

Plan du rapport

Le rapport sera organisé comme suit :

|  *Introduction aux web services*

|  *Technologies : SOAP, WSDL, UDDI, JAXWS*

|  *Cas Pratique*

|  *Conclusion*

I. Introduction aux 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).



Schéma d'appel d'un web service

Un web service :

- ✓ Composant d'une application qui offre un service.
- ✓ Permet la communication entre les applications.
- ✓ Basé sur la technologie XML pour la représentation des données.
- ✓ Basé généralement sur HTTP pour la transmission des messages.

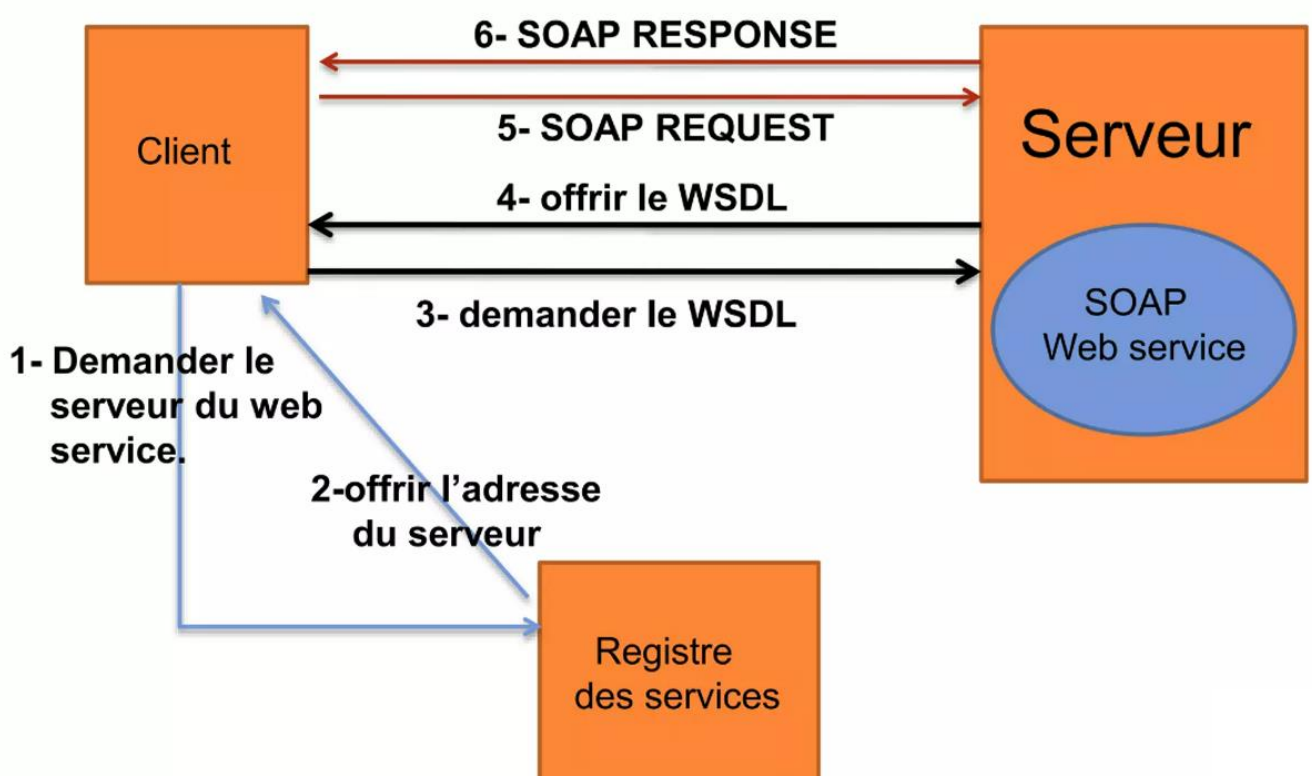
II. Les technologies

SOAP (Simple Object Access Protocol) est un protocole de communication utilisé pour les web services. Il est basé sur XML et permet de transmettre des messages structurés entre des applications. SOAP est un protocole standard, soutenu par de nombreux langages de programmation et de frameworks.

WSDL (Web Services Description Language) est un langage de description de services web. Il permet de décrire les interfaces de services web, ainsi que leurs méthodes et leurs paramètres. WSDL est un langage standard, soutenu par de nombreux langages de programmation et de frameworks.

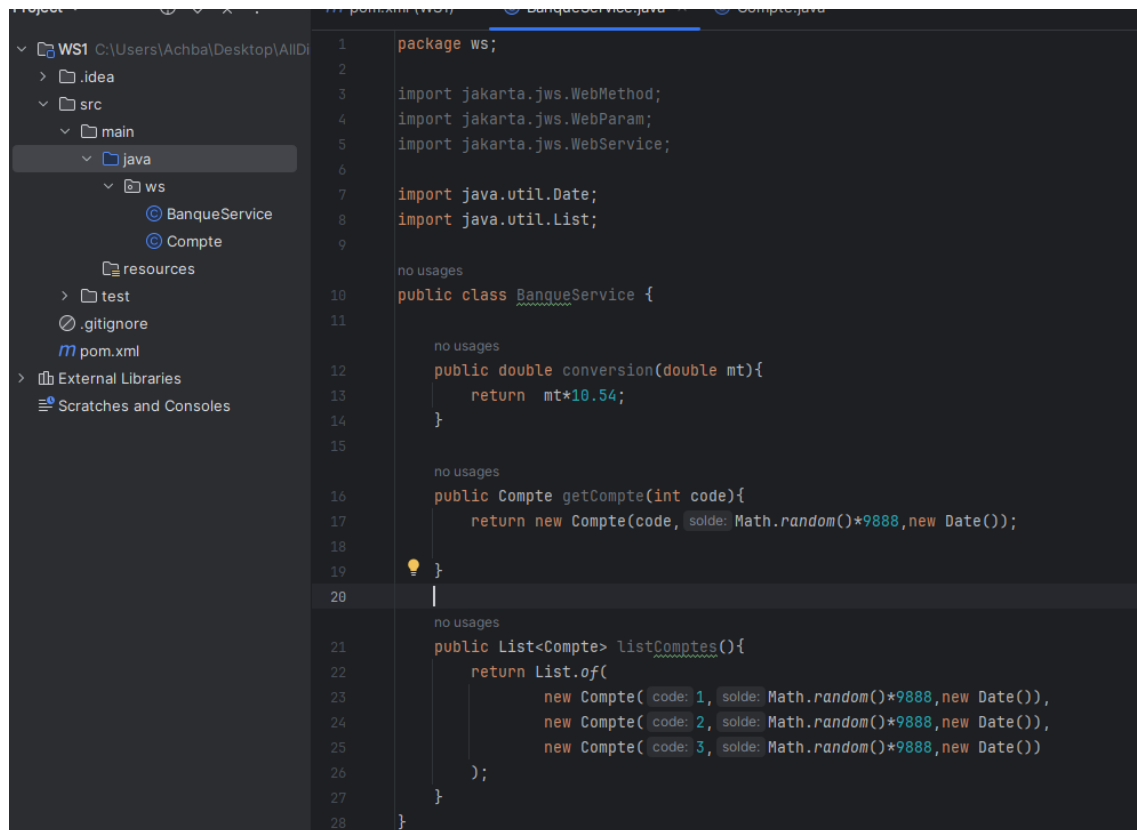
UDDI (Universal Description, Discovery, and Integration) est un registre de services web. Il permet aux développeurs de trouver des services web disponibles. UDDI est un service public, soutenu par de nombreux fournisseurs.

JAXWS (Java API for XML Web Services) est une API Java pour les web services SOAP. Elle permet aux développeurs de créer et d'utiliser des web services SOAP en Java. JAXWS est une API standard, soutenue par de nombreux IDE Java.



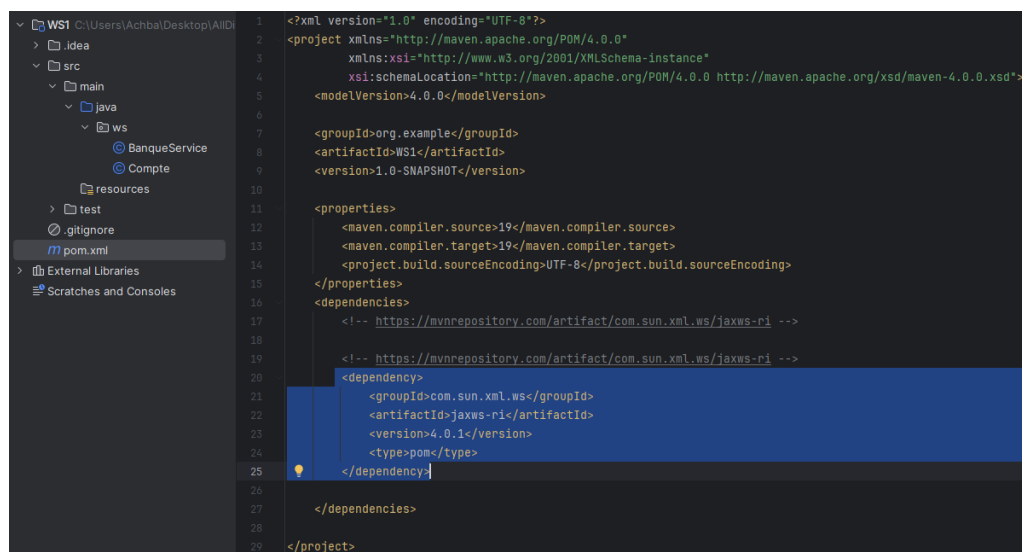
III. Cas pratique

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



```
1 package ws;
2
3 import jakarta.jws.WebMethod;
4 import jakarta.jws.WebParam;
5 import jakarta.jws.WebService;
6
7 import java.util.Date;
8 import java.util.List;
9
10 no usages
11 public class BanqueService {
12
13     no usages
14     public double conversion(double mt){
15         return mt*10.54;
16     }
17
18     no usages
19     public Compte getCompte(int code){
20         return new Compte(code, solde: Math.random()*9888,new Date());
21     }
22
23     no usages
24     public List<Compte> listComptes(){
25         return List.of(
26             new Compte( code: 1, solde: Math.random()*9888,new Date()),
27             new Compte( code: 2, solde: Math.random()*9888,new Date()),
28             new Compte( code: 3, solde: Math.random()*9888,new Date())
29         );
30     }
31 }
```

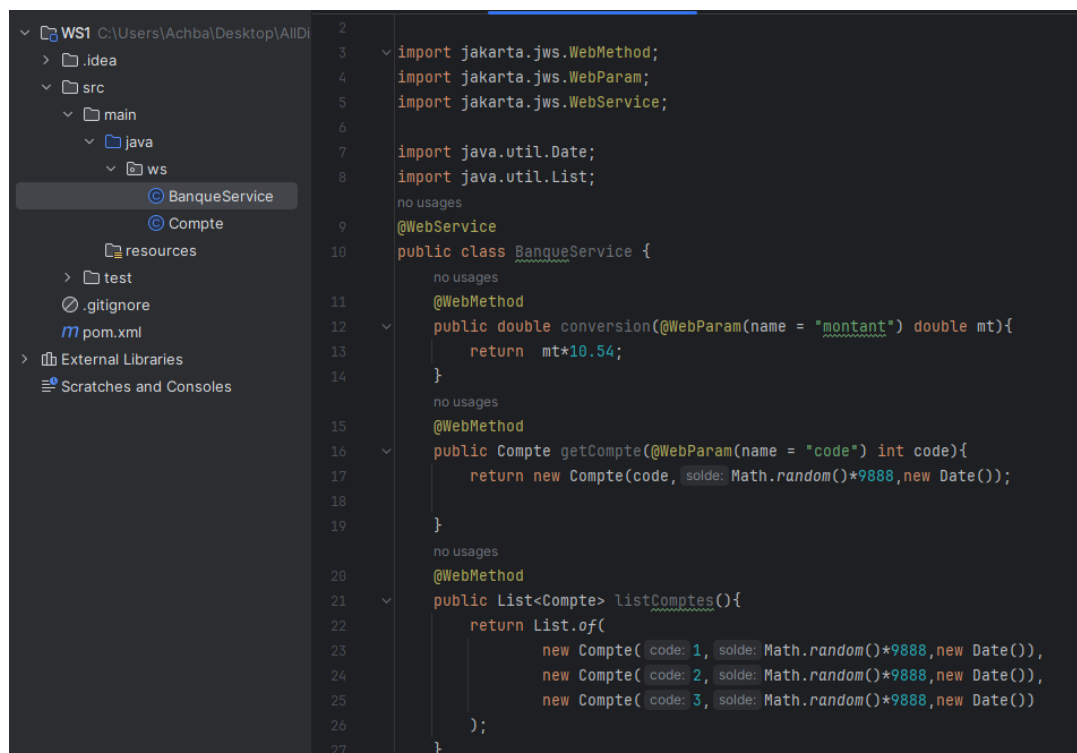
Pour la rendre un web service , il nous faut donc d'abord ajouter une dépendance de JAX-WS qui va nous permettre d'ajouter les annotations adéquates à la classe.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>org.example</groupId>
8     <artifactId>WS1</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>19</maven.compiler.source>
13         <maven.compiler.target>19</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16     <dependencies>
17         <!-- https://mvnrepository.com/artifact/com.sun.xml.ws/jaxws-ri -->
18         <!-- https://mvnrepository.com/artifact/com.sun.xml.ws/jaxws-ri -->
19         <dependency>
20             <groupId>com.sun.xml.ws</groupId>
21             <artifactId>jaxws-ri</artifactId>
22             <version>4.0.1</version>
23             <type>pom</type>
24         </dependency>
25     </dependencies>
26
27 </project>
```

Notre webservice est donc comme suivant :

- **@WebService** : Cette annotation est utilisée pour marquer une classe en tant que classe de service web. Elle indique que la classe contient des opérations qui peuvent être exposées en tant que services web.
- **@WebParam** : Cette annotation est utilisée pour annoter les paramètres des méthodes d'un service web. Elle permet de spécifier des informations supplémentaires sur les paramètres, telles que leur nom dans le service web publié.
- **@WebMethod** : Cette annotation est utilisée pour spécifier qu'une méthode d'une classe annotée avec @WebService est une opération qui doit être exposée en tant que service web. Par défaut, toutes les méthodes publiques de la classe de service web sont exposées, mais vous pouvez utiliser @WebMethod pour contrôler explicitement quelles méthodes doivent être exposées.



```
1  2
3  import jakarta.jws.WebMethod;
4  import jakarta.jws.WebParam;
5  import jakarta.jws.WebService;
6
7  import java.util.Date;
8  import java.util.List;
9
10 no usages
11 @WebService
12 public class BanqueService {
13     no usages
14     @WebMethod
15     public double conversion(@WebParam(name = "montant") double mt){
16         return mt*10.54;
17     }
18     no usages
19     @WebMethod
20     public Compte getCompte(@WebParam(name = "code") int code){
21         return new Compte(code, solde: Math.random()*9888,new Date());
22     }
23     no usages
24     @WebMethod
25     public List<Compte> listComptes(){
26         return List.of(
27             new Compte( code: 1, solde: Math.random()*9888,new Date()),
28             new Compte( code: 2, solde: Math.random()*9888,new Date()),
29             new Compte( code: 3, solde: Math.random()*9888,new Date())
30         );
31     }
32 }
```

Le déploiement d'un service web, consiste à rendre ce service accessible sur un serveur web de manière à ce que d'autres applications puissent le consommer. Le déploiement d'un service web est une étape essentielle pour le mettre en ligne et permettre son utilisation par des clients.

Pour configurer le serveur web pour prendre en charge le service web, on va créer la classe `ServerJWS` qui contient la méthode **Endpoint.publish()**, qui est une méthode fournie par l'API JAX-WS (Java API for XML Web Services) de Java qui permet de publier un service web SOAP (Simple Object Access Protocol) en tant que point de terminaison sur un serveur.. Autrement dit, la méthode permet de démarrer un serveur http pour consommer le webservice BanqueService.

```

1  import jakarta.xml.ws.Endpoint;
2  import ws.BanqueService;
3
4  public class ServerJWS {
5      public static void main(String[] args) {
6          Endpoint.publish( address: "http://0.0.0.0:4455/", new BanqueService());
7          System.out.println("Web service déployé sur http://0.0.0.0:4455/");
8      }
9  }
10
Run  ServerJWS x
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
Web service déployé sur http://0.0.0.0:4455/

```

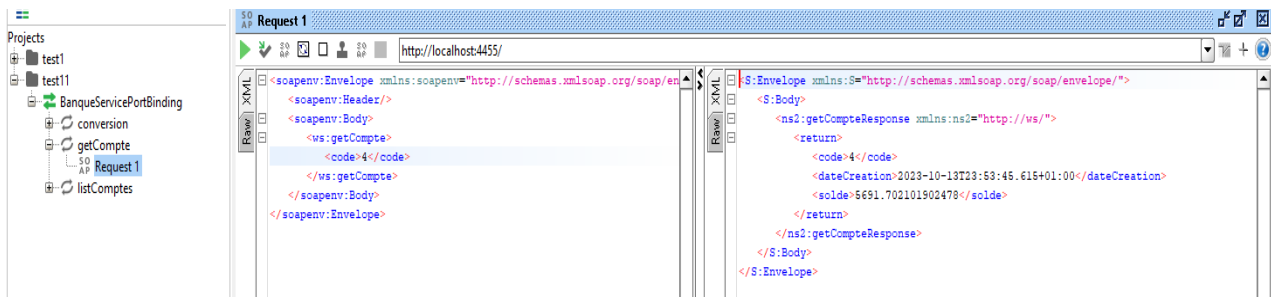
Le server est démarré on peut avoir donc notre WSDL :

```

<?xml version='1.0'?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/08/oasis-200408-wss-wssecurity-utility-1.0.xsd" xmlns:ws="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
name="BanqueService" targetNamespace="http://ws/">
  <types>
    <xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      base="http://ws/"
      schemaLocation="http://localhost:4455/?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="conversion">
    <part name="parameters" element="tns:conversion"/>
  </message>
  <message name="conversionResponse">
    <part name="parameters" element="tns:conversionResponse"/>
  </message>
  <message name="getCompte">
    <part name="parameters" element="tns:getCompte"/>
  </message>
  <message name="getCompteResponse">
    <part name="parameters" element="tns:getCompteResponse"/>
  </message>
  <message name="listComptes">
    <part name="parameters" element="tns:listComptes"/>
  </message>
  <message name="listComptesResponse">
    <part name="parameters" element="tns:listComptesResponse"/>
  </message>
  <portType name="BanqueService">
    <operation name="conversion">
      <input wsam:Action="http://ws/BanqueService/conversionRequest" message="tns:conversion"/>
      <output wsam:Action="http://ws/BanqueService/conversionResponse" message="tns:conversionResponse"/>
    </operation>
    <operation name="getCompte">
      <input wsam:Action="http://ws/BanqueService/getCompteRequest" message="tns:getCompte"/>
      <output wsam:Action="http://ws/BanqueService/getCompteResponse" message="tns:getCompteResponse"/>
    </operation>
    <operation name="listComptes">
      <input wsam:Action="http://ws/BanqueService/listComptesRequest" message="tns:listComptes"/>
      <output wsam:Action="http://ws/BanqueService/listComptesResponse" message="tns:listComptesResponse"/>
    </operation>
  </portType>
  <binding name="BanqueServicePortBinding" type="tns:BanqueService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="conversion">
      <soap:operation soapAction="">
        <input
          <soap:body use="literal"/>
        </input>
        <output
          <soap:body use="literal"/>
        </output>
      </operation>
    <operation name="getCompte">
      <soap:operation soapAction="">
        <input
          <soap:body use="literal"/>
        </input>
        <output
          <soap:body use="literal"/>
        </output>
      </operation>
    <operation name="listComptes">
      <soap:operation soapAction="">
        <input
          <soap:body use="literal"/>
        </input>
        <output
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
  </definitions>

```


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).

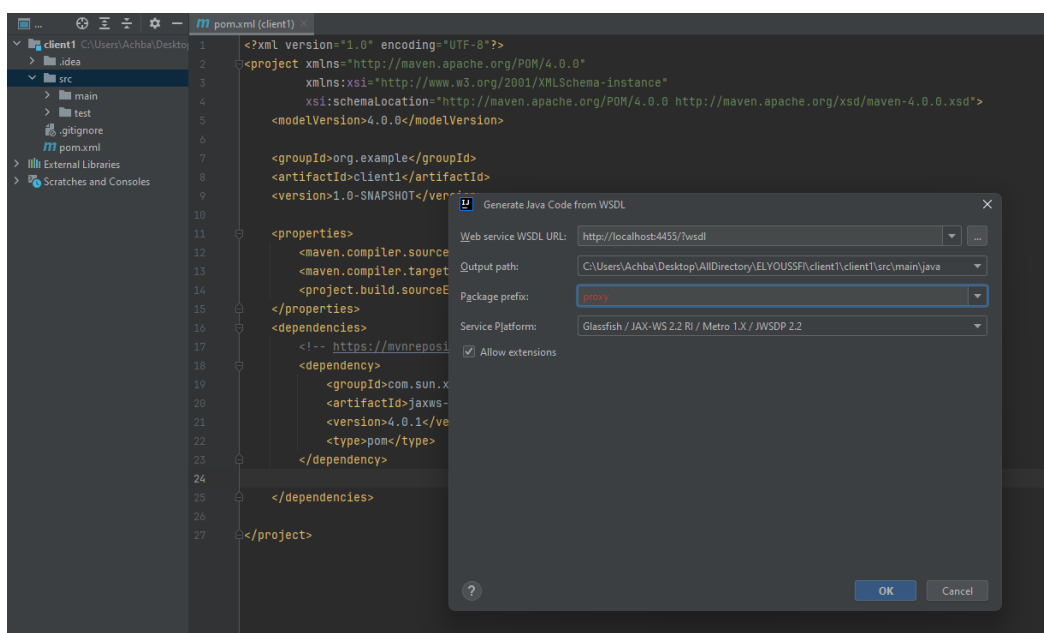


Pour créer un client Java et générer un proxy pour consommer un service web, nous allons utiliser JAX-WS (Java API for XML Web Services).

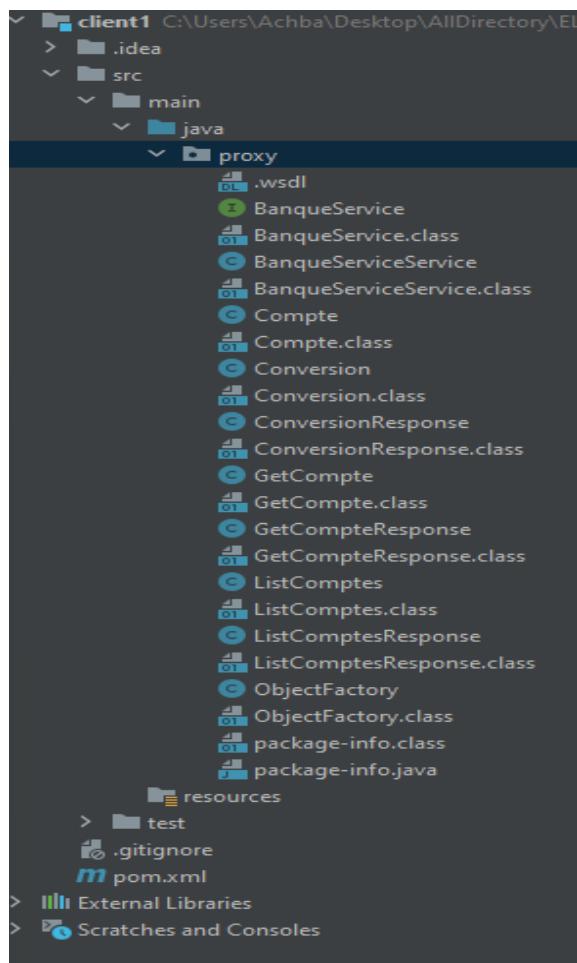
Voici les étapes à suivre :

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

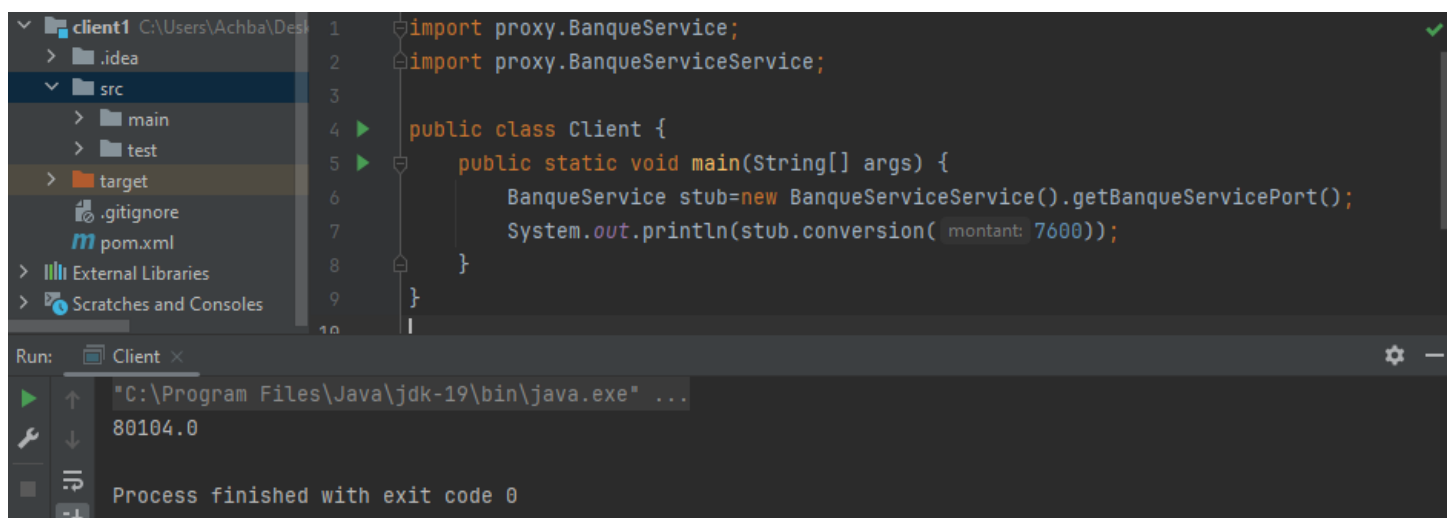
Etape 1 :



Etape 2 :



Etape 3&4 :



Conclusion :

Pour consommer un service web en Java, nous utilisons JAX-WS (Java API for XML Web Services). Dans IntelliJ IDEA, nous générons un proxy à partir du fichier WSDL du service, ce qui produit automatiquement des classes Java, y compris un Stub, qui agit comme une passerelle entre notre application cliente et le service distant. Le Stub gère la sérialisation et la désérialisation des données pour la communication. Ensuite, dans notre classe cliente Java, nous utilisons ce Stub pour appeler les méthodes du service web distant. Le STUB communique avec le Skeleton côté serveur. Le Skeleton est un composant qui réside sur le serveur et sert d'intermédiaire entre le service web distant et l'objet distant réel qui implémente le service. Lorsqu'une requête est reçue par le Skeleton, il transfère la requête à l'objet distant, qui effectue le traitement nécessaire. Une fois que l'objet distant a produit une réponse, le Skeleton renvoie la réponse au Stub, qui la renvoie ensuite au client. Le Skeleton est essentiel pour gérer la communication entre le client et le service distant du côté serveur, permettant ainsi au client d'appeler les méthodes du service web de manière transparente.

