

Master Intelligence Artificielle et Analyse des Données

Systèmes Distribués

COMPTE RENDU DU TP5

Réalisé par :

HALIMA DAOUDI

Année universitaire : 2023 – 2024

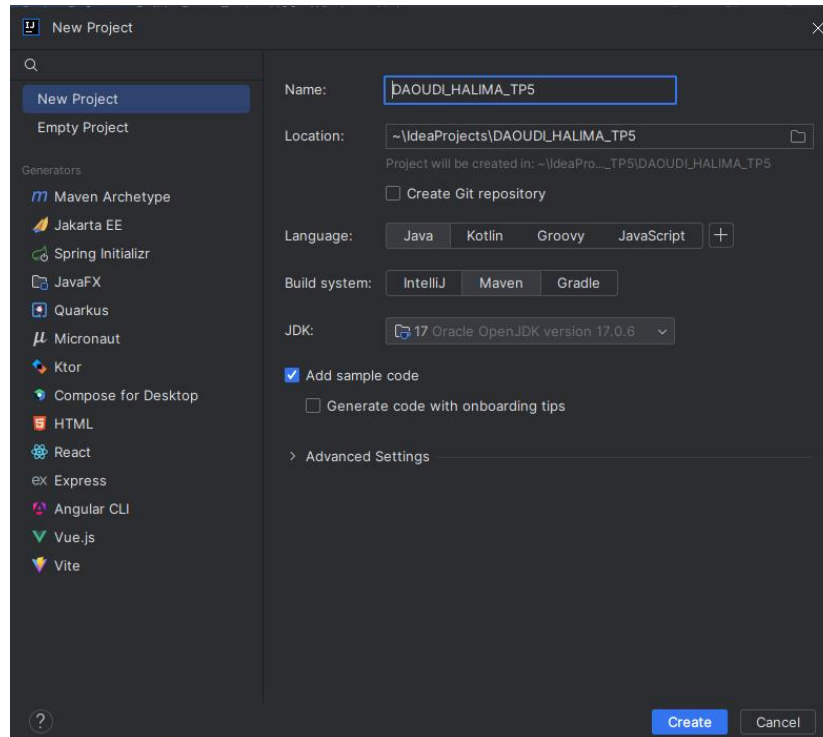
INTRODUCTION

Dans ce TP, je vais explorer la création et le déploiement d'un service web SOAP, ainsi que l'interaction avec ce service à l'aide d'un client SOAP Java. Les services web SOAP permettent l'échange de données structurées entre applications sur différents systèmes d'exploitation et langages de programmation. SOAP utilise des protocoles standards comme HTTP et XML pour garantir l'interopérabilité entre les systèmes.

Voici les étapes que je réalise dans ce TP :

1. Créer un Web service qui permet de :
 - Convertir un montant de l'euro en dirhams marocains (DH)
 - Consulter un compte
 - Consulter une liste de comptes
2. Déployer le Web service avec un simple serveur JaxWS.
3. Consulter et analyser le WSDL avec un navigateur HTTP.
4. Tester les opérations du Web service avec un outil comme SoapUI ou Oxygen.
5. Créer un client SOAP Java :
 - Générer le stub à partir du WSDL.
 - Créer un client SOAP pour le Web service.

Je vais commencer par créer un projet pour étudier le rôle des protocoles SOAP et WSDL dans le développement d'applications distribuées.



Etape 1 : Créer un Web service

Je commence par créer un package nommé ws et y définir une classe Compte, qui inclura des variables pour le code du compte, le solde, et la date de création. Je m'assure également d'ajouter des méthodes getters et setters pour permettre la manipulation et l'accès sécurisé à ces propriétés.

```
package ws;
import java.time.LocalDate;

7 usages
public class Compte {
    3 usages
    private int code;
    3 usages
    private double solde;
    3 usages
    private LocalDate dateCreation;
    no usages
    public Compte() {
    }
    4 usages
    public Compte(int code, double solde, LocalDate dateCreation) {
        this.code = code;
        this.solde = solde;
        this.dateCreation = dateCreation;
    }
}
```

```

    public int getCode() {
        return code;
    }
    no usages
    public void setCode(int code) {
        this.code = code;
    }
    no usages
    public double getSolde() {
        return solde;
    }
    no usages
    public void setSolde(double solde) {
        this.solde = solde;
    }
    no usages
    public LocalDate getDateCreation() {
        return dateCreation;
    }
    no usages
    public void setDateCreation(LocalDate dateCreation) {
        this.dateCreation = dateCreation;
    }
}

```

Je commence par ajouter une dépendance dans le fichier pom.xml pour intégrer JAX-WS, qui est nécessaire pour le développement de services web SOAP en Java.

```

<dependencies>
    <dependency>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-ri</artifactId>
        <version>4.0.2</version>
        <type>pom</type>
    </dependency>
</dependencies>

```

Je crée un service web nommé BanqueService dans le package ws. Ce service offre des méthodes pour convertir des euros en dirhams marocains, récupérer les détails d'un compte bancaire spécifique par son code, et lister plusieurs comptes avec leurs détails. Ce service utilise des annotations pour intégrer SOAP et rendre les méthodes accessibles via un réseau.

```

package ws;

import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebService;
import java.util.Date;
import java.util.List;
import java.lang.Math;

@WebService(serviceName = "BanqueWS")
public class BanqueService {
    @WebMethod(operationName = "conversionEuroToDH")
    public double conversion(@WebParam(name = "montant") double mt) {
        return mt*11.3;
    }

    @WebMethod
    public Compte getCompte(@WebParam(name = "code") int code){
        return new Compte(code, Math.random()*80000, new Date());
    }

    @WebMethod
    public List<Compte> listComptes(){
        return List.of(
            new Compte(1, Math.random()*80000, new Date()),
            new Compte(2, Math.random()*80000, new Date()),
            new Compte(3, Math.random()*80000, new Date())
        );
    }
}

```

Etape 2 : Déployer le Web service avec un simple Serveur JaxWS

je déploie le service web créé précédemment en utilisant un serveur JAX-WS simple. Je crée une classe ServerJWS avec une méthode principale qui publie le service BanqueService à une URL spécifique. Ce processus rend le service web accessible sur le réseau local à l'adresse définie, permettant ainsi aux clients de consommer les services offerts par BanqueService.

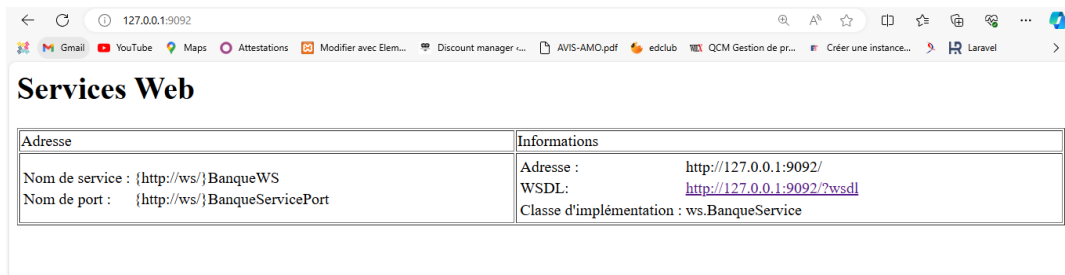
```

public class ServerJWS {
    public static void main(String[] args) {
        String url = "http://127.0.0.1:9092/";
        Endpoint.publish(url, new BanqueService());
        System.out.println("Web service deployed on " + url);
    }
}

```

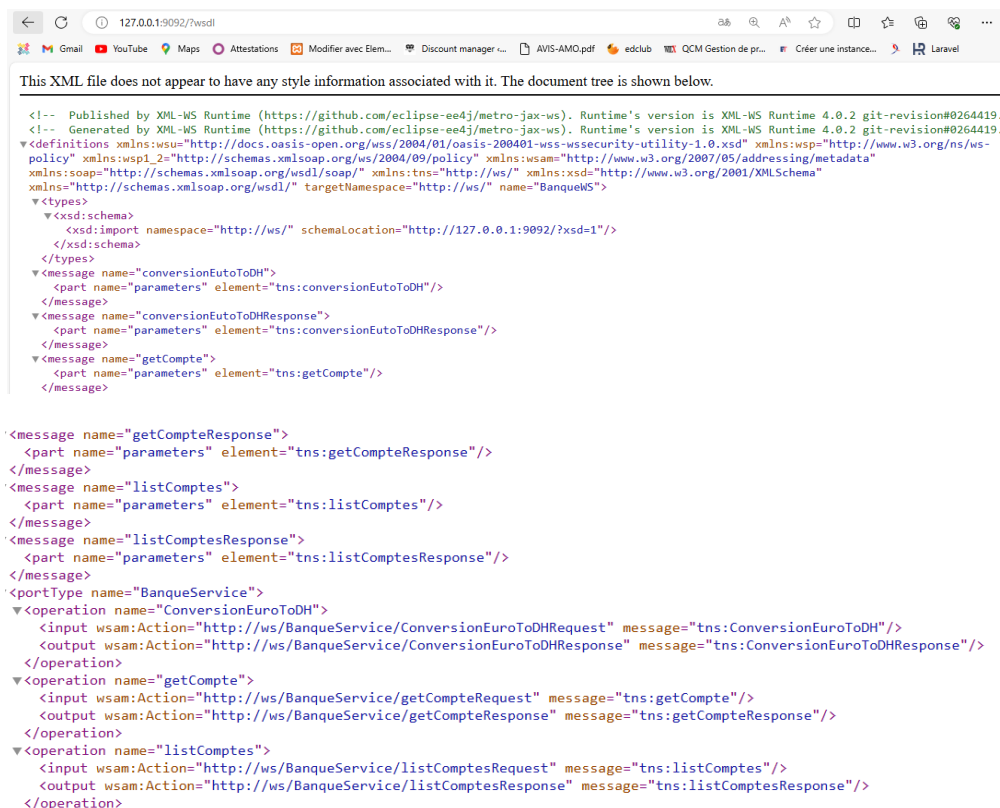
Etape 3 : Consulter et analyser le WSDL avec un Browser http

Je consulte le service web que j'ai déployé pour vérifier ses détails et sa disponibilité. En accédant à l'adresse spécifiée, je peux voir des informations telles que l'adresse du service, le nom du service (BanqueWS), le nom du port (BanqueServicePort), et l'adresse du fichier WSDL qui décrit les opérations disponibles et la structure des messages échangés. Cela me permet de confirmer que le service est actif et prêt à être utilisé par des clients.



Adresse	Informations
Nom de service : {http://ws/}BanqueWS	Adresse : http://127.0.0.1:9092/
Nom de port : {http://ws/}BanqueServicePort	WSDL: http://127.0.0.1:9092/?wsdl
	Classe d'implémentation : ws.BanqueService

Je consulte le lien <http://127.0.0.1:9092/?wsdl> qui me permet d'accéder au fichier WSDL de mon service web BanqueService. Ce fichier fournit des détails sur les opérations disponibles, les formats de message et les protocoles nécessaires pour interagir avec le service. Il est essentiel pour configurer des clients ou d'autres services afin de faciliter une communication efficace avec mon service.

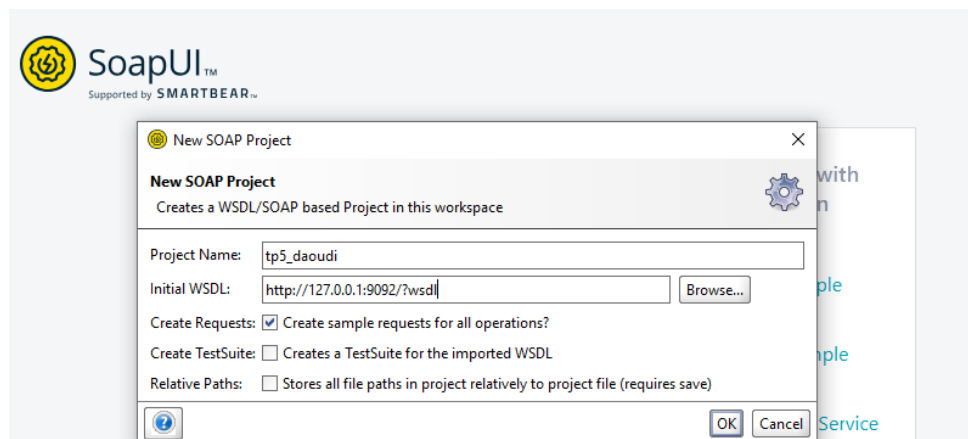


This XML file does not appear to have any style information associated with it. The document tree is shown below.

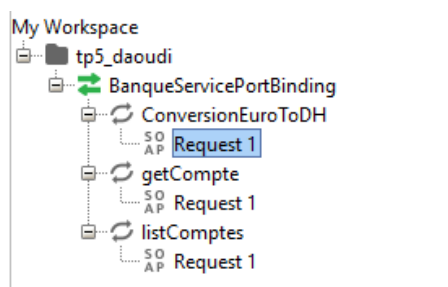
```
<!-- Published by XML-WS Runtime (https://github.com/eclipse-ee4j/metro-jax-ws). Runtime's version is XML-WS Runtime 4.0.2 git-revision#0264419.
<!-- Generated by XML-WS Runtime (https://github.com/eclipse-ee4j/metro-jax-ws). Runtime's version is XML-WS Runtime 4.0.2 git-revision#0264419.
<?xml version="1.0" encoding="UTF-8" ?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-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" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://ws/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws/" name="BanqueWS">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws/" schemaLocation="http://127.0.0.1:9092/?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="conversionEuroToDH">
    <part name="parameters" element="tns:conversionEuroToDH"/>
  </message>
  <message name="conversionEuroToDHResponse">
    <part name="parameters" element="tns:conversionEuroToDHResponse"/>
  </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="ConversionEuroToDH">
      <input wsam:Action="http://ws/BanqueService/ConversionEuroToDHRequest" message="tns:ConversionEuroToDH"/>
      <output wsam:Action="http://ws/BanqueService/ConversionEuroToDHResponse" message="tns:ConversionEuroToDHResponse"/>
    </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>
</definitions>
```

Etape 4 : Tester les opérations du web service avec un outil comme SoapUI ou Oxygen

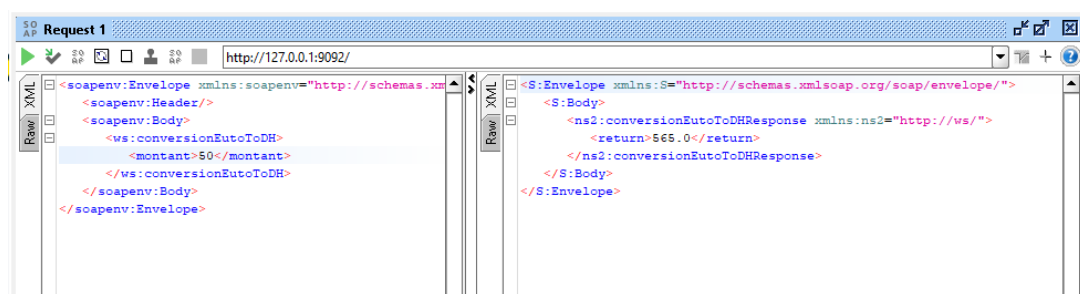
Je teste les opérations de mon service web en utilisant SoapUI. Je crée un nouveau projet SOAP, en entrant l'URL du WSDL de mon service et en générant des requêtes d'exemple pour toutes les opérations. Cela me permet d'interagir facilement avec mon service et de vérifier son fonctionnement.



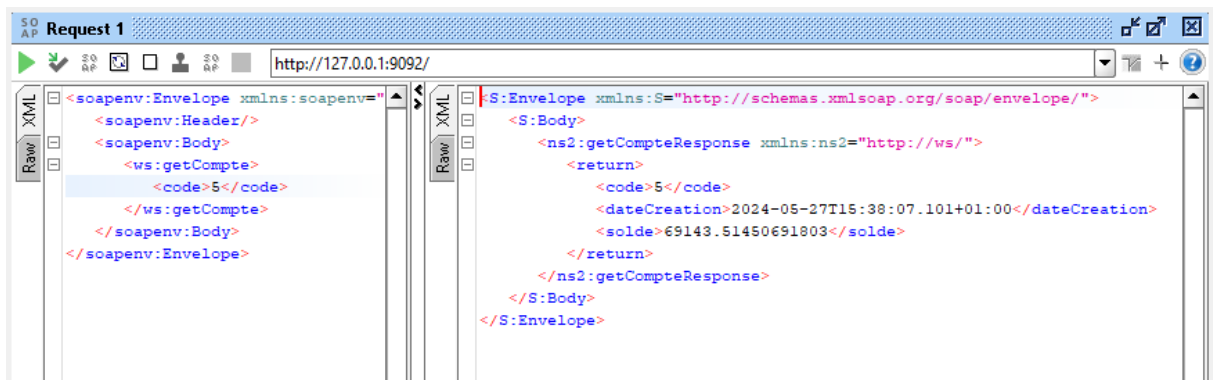
Dans mon projet SoapUI tp5_daoudi, je visualise les différentes opérations de mon service web BanqueServicePortBinding, telles que ConversionEuroToDH, getCompte, et listComptes. Chaque opération est équipée d'une requête prédéfinie, permettant de tester facilement le fonctionnement de chaque méthode du service.



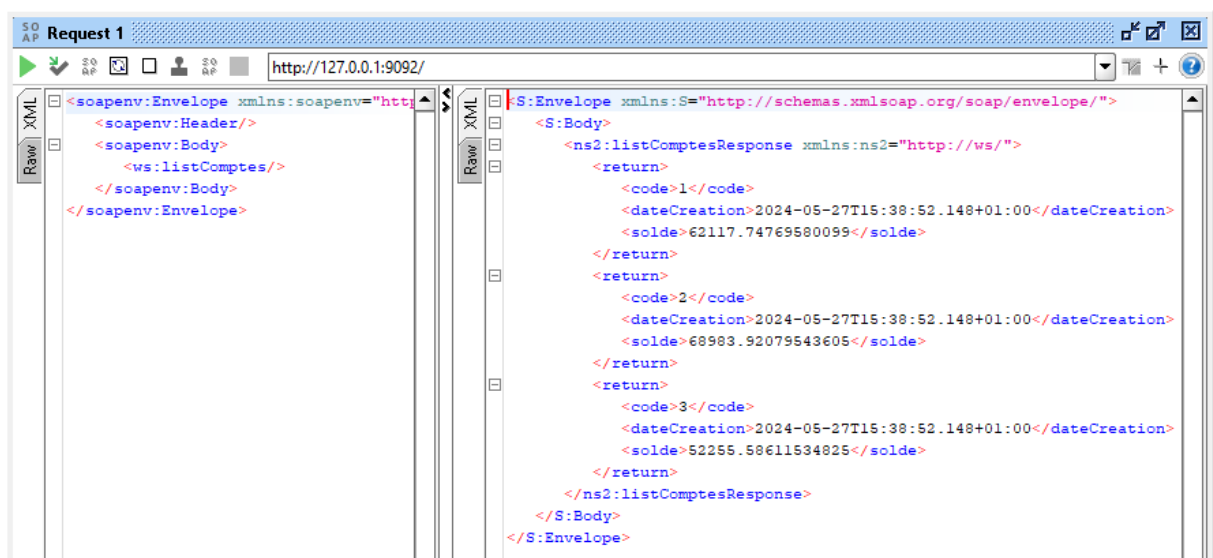
Je teste l'opération ConversionEuroToDH en envoyant 50 euros et je reçois en réponse 565.0 dirhams, confirmant ainsi le bon fonctionnement du service web.



J'envoie une requête pour récupérer les informations du compte avec le code 5 et je reçois en réponse les détails complets du compte, confirmant le bon fonctionnement de l'opération getCompte.

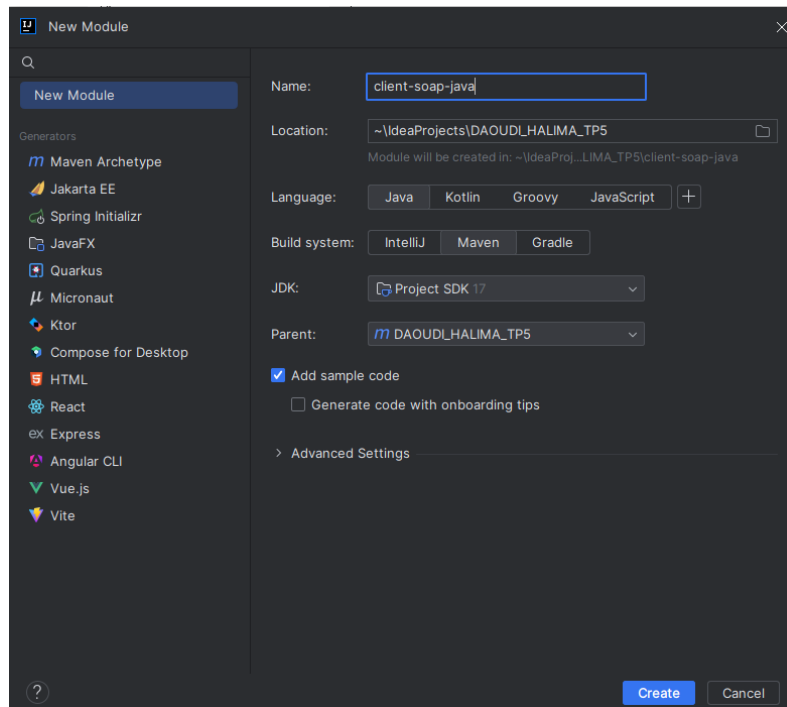


J'envoie une requête pour lister tous les comptes disponibles et je reçois en réponse les détails de plusieurs comptes, y compris leur code, date de création, et solde. Cette réponse confirme que l'opération fonctionne correctement en récupérant une liste des comptes existants.

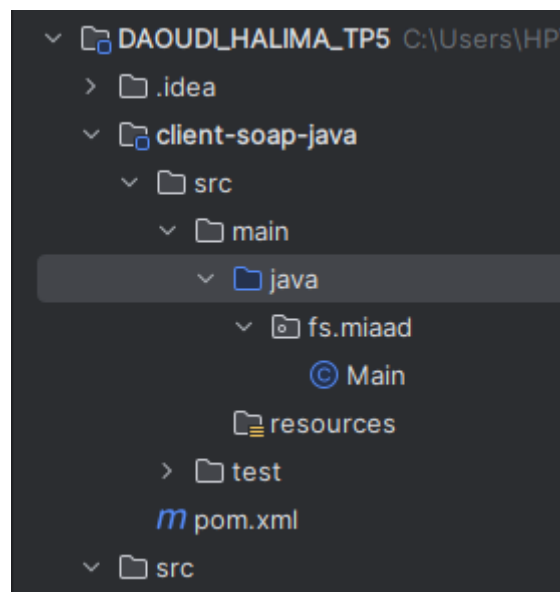


5. Créer un Client SOAP Java

J'ajoute un nouveau module à mon projet IntelliJ. Je nomme le module client-soap-java et je le configure pour utiliser Java avec le système de build Maven, en choisissant JDK 17. En ajoutant ce module, je peux développer et tester des interactions client avec mon service web SOAP de manière isolée et organisée.



Voici l'architecture du module client-soap-java dans mon projet :

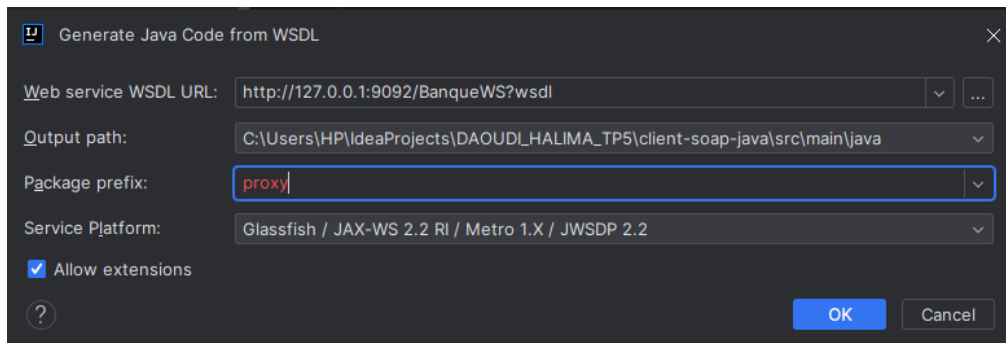


En pom.xml, j'ajoute la dépendance suivante pour intégrer JAX-WS :

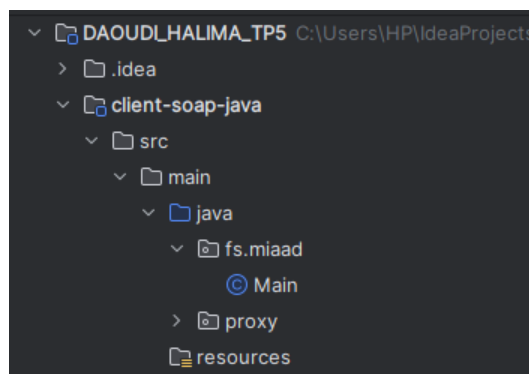
```
<dependencies>
  <dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-ri</artifactId>
    <version>4.0.2</version>
    <type>pom</type>
  </dependency>
</dependencies>
```

- Générer le Stub à partir du WSDL :

Je génère le code Java à partir du fichier WSDL en utilisant l'outil intégré d'IntelliJ IDEA. Je spécifie l'URL du WSDL de mon service web, le chemin de sortie pour les fichiers générés, et le préfixe de package proxy. Cela crée automatiquement les classes nécessaires pour interagir avec le service web.



Après avoir généré le stub à partir du WSDL, le code généré est placé dans le package proxy de mon projet.



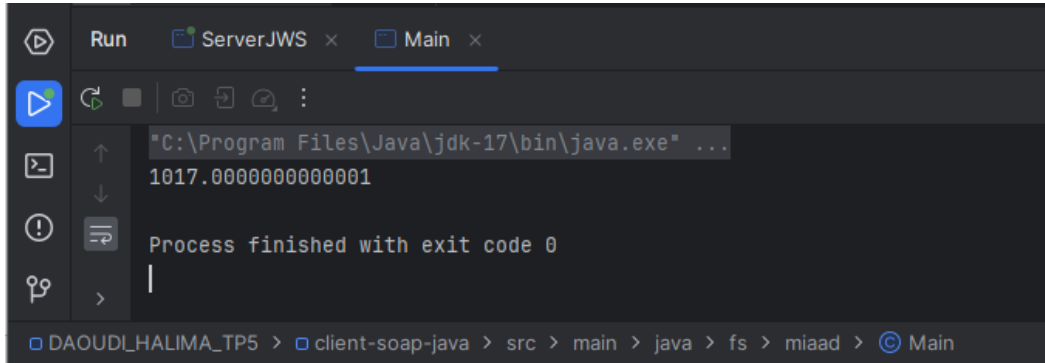
- Créer un client SOAP pour le web service

Je crée un client SOAP pour le service web en important les classes générées dans le package proxy. Dans la classe Main, j'initialise le service BanqueService et j'appelle la méthode conversionEuroToDH pour tester la conversion d'un montant en euros vers les dirhams, affichant le résultat dans la console.

```
package fs.miaad;
import proxy.BanqueService;
import proxy.BanqueWS;

public class Main {
    public static void main(String[] args) {
        BanqueService proxy = new BanqueWS().getBanqueServicePort();
        System.out.println(proxy.conversionEuroToDH(montant: 90));
    }
}
```

J'exécute le client SOAP, et le programme affiche 1017.0, confirmant que la conversion de 90 euros en dirhams fonctionne correctement.



```
Run ServerJWS x Main x
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
1017.0000000000001
Process finished with exit code 0
|
DAOUDI_HALIMA_TP5 > client-soap-java > src > main > java > fs > miaad > Main
```

J'ai modifié le client SOAP en ajoutant des appels pour récupérer un compte spécifique avec le code 4 et pour lister tous les comptes disponibles. Le programme affiche les détails de chaque compte, confirmant le bon fonctionnement des méthodes `getCompte` et `listComptes`.

```
package fs.miaad;
import proxy.BanqueService;
import proxy.BanqueWS;
import proxy.Compte;

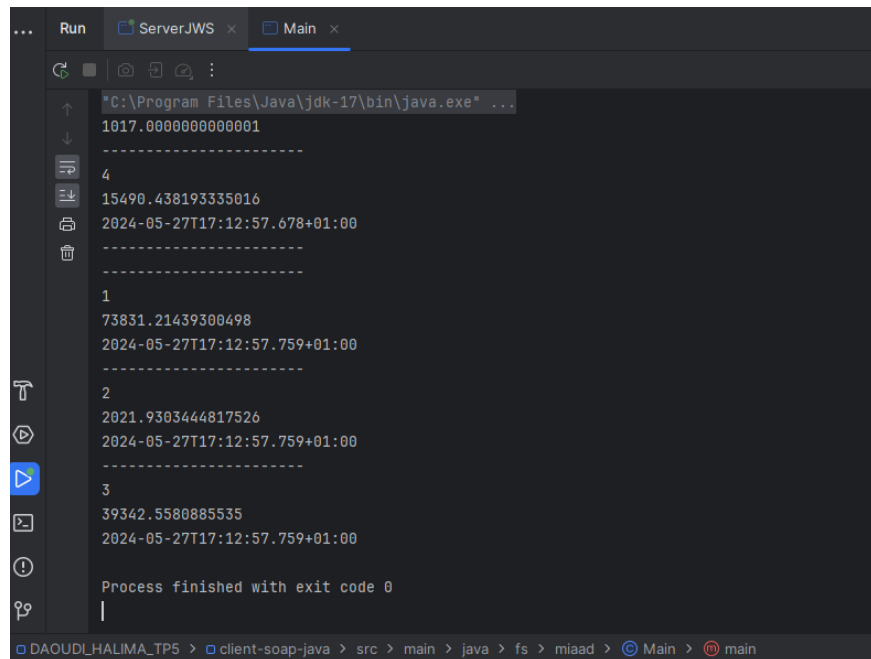
public class Main {
    public static void main(String[] args) {
        BanqueService proxy = new BanqueWS().getBanqueServicePort();
        System.out.println(proxy.conversionEuroToDH( montant: 90.0));

        System.out.println("-----");
        Compte compte = proxy.getCompte( code: 4);
        System.out.println(compte.getCode());
        System.out.println(compte.getSolde());
        System.out.println(compte.getDateCreation());

        System.out.println("-----");

        proxy.listComptes().forEach(cp -> {
            System.out.println("-----");
            System.out.println(cp.getCode());
            System.out.println(cp.getSolde());
            System.out.println(cp.getDateCreation());
        });
    }
}
```

Voici l'exécution finale de mon client SOAP. Le programme affiche la conversion de 90 euros en dirhams, les détails d'un compte spécifique avec le code 4, ainsi que la liste de tous les comptes disponibles. Cela confirme que toutes les méthodes du service web fonctionnent correctement et renvoient les résultats attendus.



```
... Run ServerJWS x Main x
C:\Program Files\Java\jdk-17\bin\java.exe" ...
1017.0000000000001
-----
4
15490.438193335016
2024-05-27T17:12:57.678+01:00
-----
1
73831.21439300498
2024-05-27T17:12:57.759+01:00
-----
2
2021.9303444817526
2024-05-27T17:12:57.759+01:00
-----
3
39342.5580885535
2024-05-27T17:12:57.759+01:00
-----
Process finished with exit code 0
|
DAOUDLHALIMA_TP5 > client-soap-java > src > main > java > fs > miaad > Main > main
```

CONCLUSION

En conclusion, ce TP m'a offert l'opportunité de concevoir et de déployer un service web SOAP, ainsi que d'interagir avec celui-ci via un client SOAP Java. À travers ce projet, j'ai créé un service permettant la conversion de montants en dirhams marocains, la consultation de comptes individuels et de listes de comptes, en utilisant un serveur JaxWS.

L'analyse du WSDL à l'aide d'un navigateur HTTP et le test des opérations avec des outils comme SoapUI ou Oxygen m'ont permis de mieux comprendre les spécificités des services web SOAP. De plus, la génération de stubs à partir du WSDL et la création d'un client SOAP Java m'ont donné une expérience pratique de bout en bout, allant du développement à l'interaction avec le service.

Ce projet m'a donc permis de renforcer mes compétences en création de services web interopérables et robustes, en utilisant des protocoles standardisés tels que HTTP et XML.