STITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES
DE SFAX

Département Technologies de Informatique

Atelier SOA -TP05

Tester un service web avec un client JSP

Objectifs

Tester un Web Service développé avec l'API JAX-WS

1. Tester les méthodes du WS via un client JSP

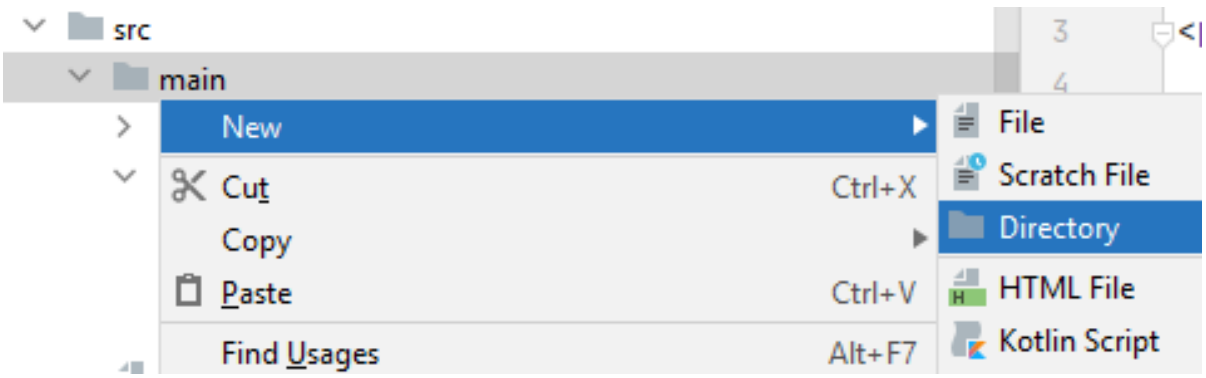
- Configurer le plugin du moteur de servlet (Jetty et Tomcat)
- Utiliser la bibliothèque JAX-WS pour générer les classes de souche
- Appeler des méthodes du service web à partir d'une page JSP

A. Créer une application web (client JSP)

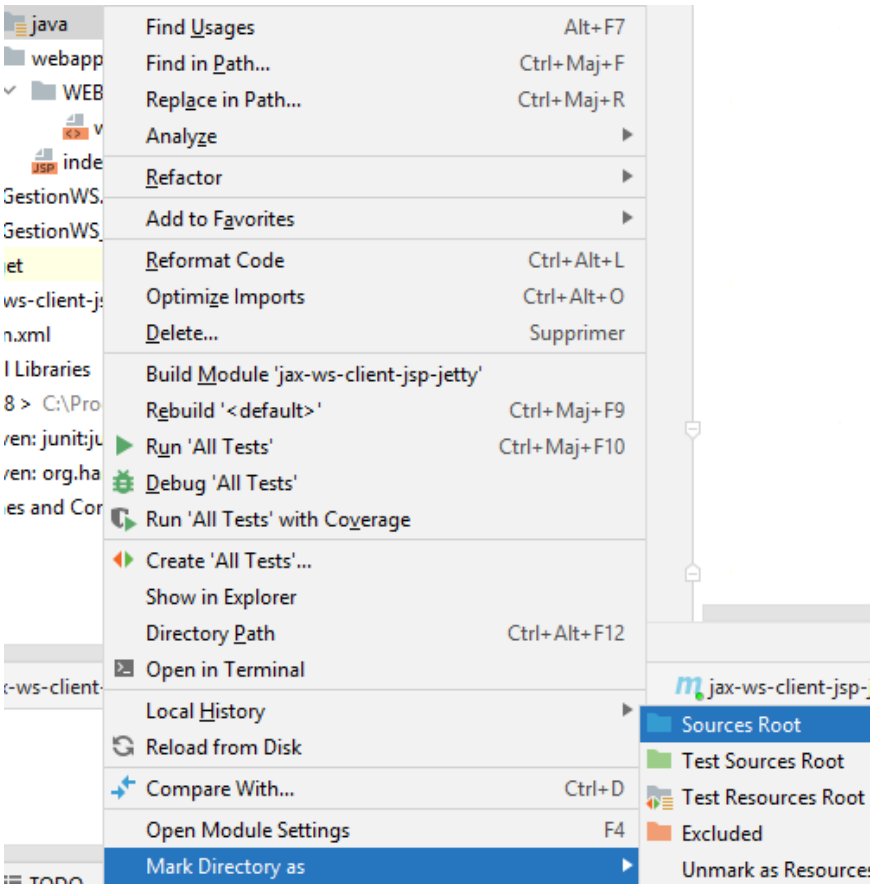
1. Créer un projet **MAVEN** (**maven-archetype-webapp**) dans **IntelliJ IDEA** nommé **jax-ws-client-jsp** (sous le dossier **workspace**) ayant les caractéristiques suivantes :

   - groupId : **org.soa.tp5**
   - artifactId : **jax-ws-client-jsp**
   - version : **1.0-SNAPSHOT**

2. Créer un dossier « **java** » sous le dossier « **main** » :



3. Rendre le dossier « **java** » nouvellement créé comme dossier de codes sources :



4. Ajouter, dans le fichier POM, la définition du plugin de **jaxws** pour générer les proxys du service Web « **GestionWS** » (directement à l'intérieur de la balise <**project**>) (voir Atelier 04 question 02)

B. Ajouter le plugin Jetty à IntelliJ IDEA

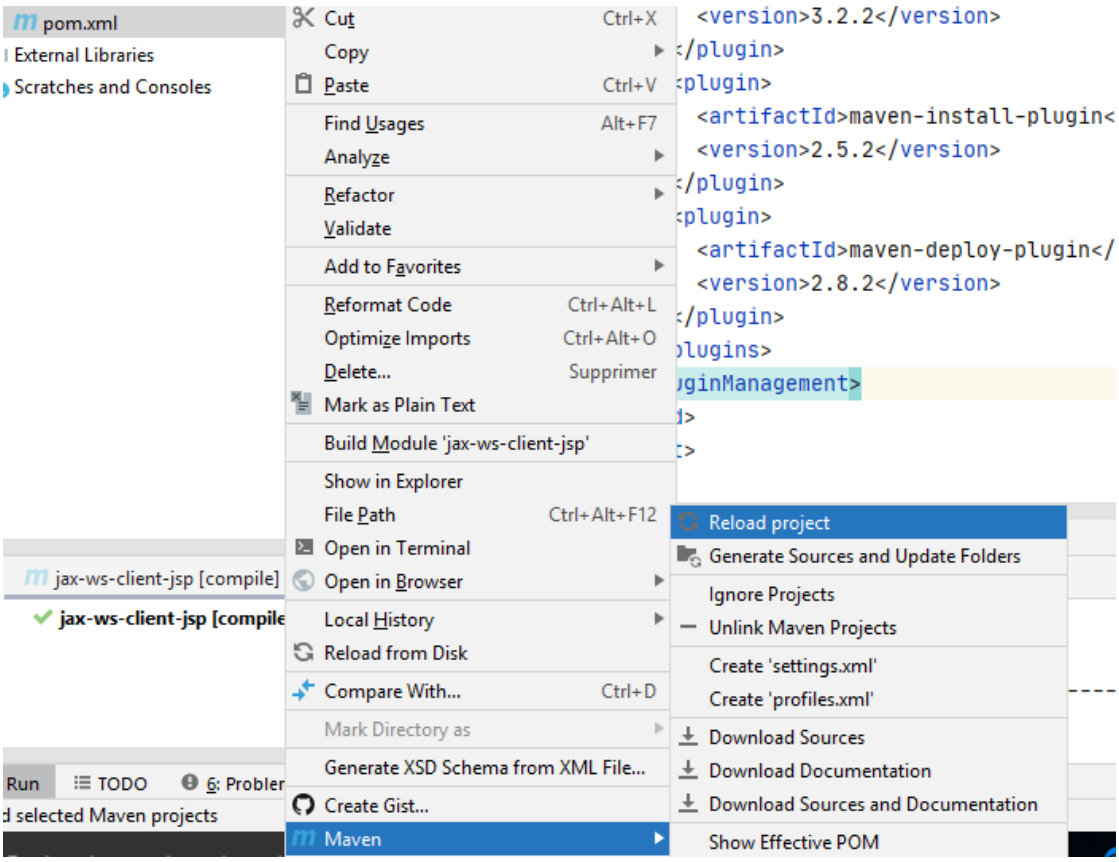
5. Editer le fichier « **pom.xml** » du projet pour ajouter le plugin du « Jetty » (moteur de servlet analogue à Tomcat et GlassFish)
6. Ajouter les lignes en gras suivantes :

```
<build>

  <plugins>
    <!-- Jetty Plugin. Default port is 8080 -->
    <plugin>
      <groupId>org.eclipse.jetty</groupId>
```

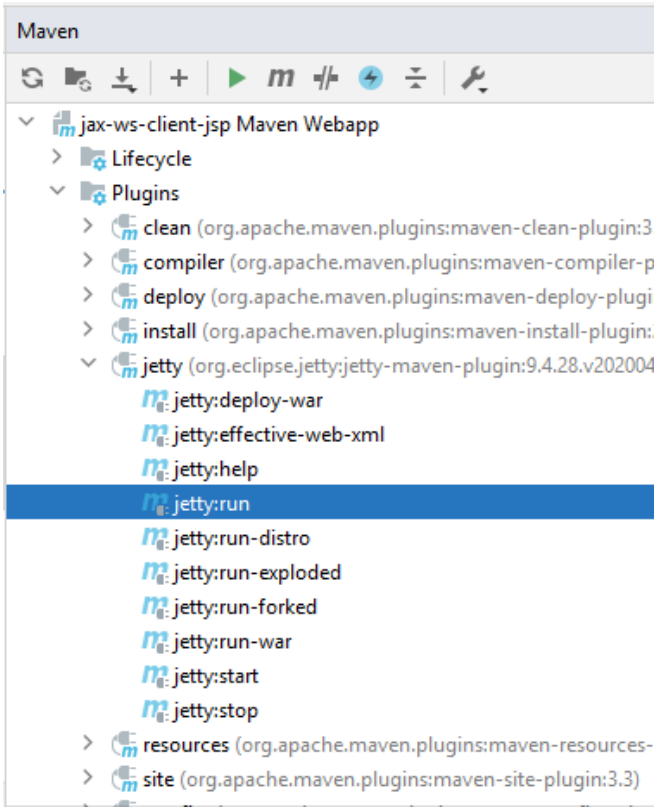
```
      <artifactId>jetty-maven-plugin</artifactId>
      <version>11.0.16</version>
      <configuration>
        <httpConnector>
          <!--host>localhost</host-->
          <port>9999</port>
        </httpConnector>
      </configuration>
    </plugin>
  </plugins>
```

```
<finalName>jax-ws-client-jsp-jetty</finalName>
```

7. Pour télécharger le plugin de Jetty, sélectionner le fichier « pom.xml » dans le volet « **Project** » puis cliquer sur la souris avec le bouton droit pour choisir la commande « **Maven/ Reload Project** » :



8. Dans le volet « **Maven** » double cliquer sue « **jetty:run** » :



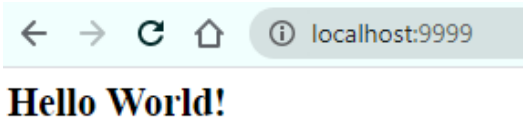
9. Si tout va bien, vous obtenez, au niveau de la console, le message suivant :

Started Jetty Server

10. Le serveur Jetty est démarré, lancer l'url suivante pour le tester :

http://localhost:9999/

11. Si tout va bien , vous recevez le résultat suivant :



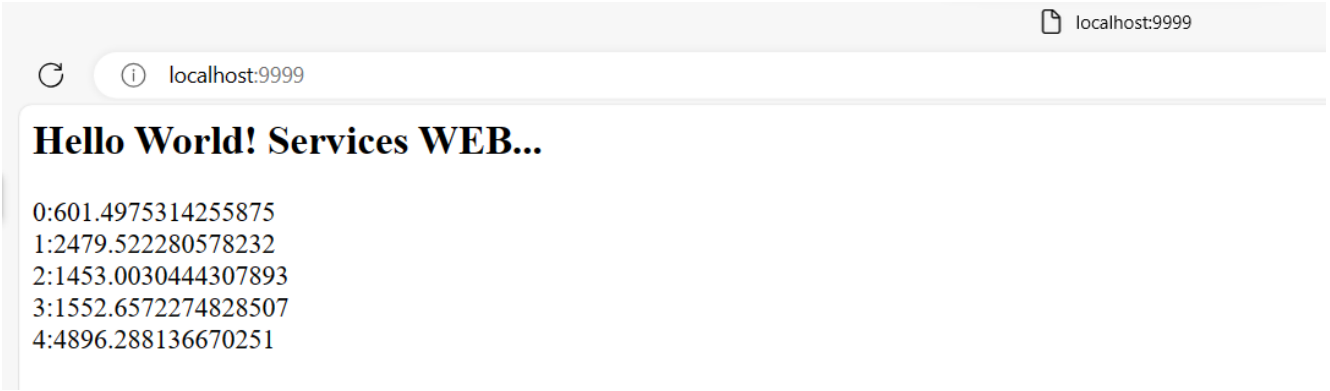
12. Ce message affiché est l'exécution de la page « **index.jsp** ». Modifier, maintenant, cette page pour invoquer le service web « **GesionWS** » et lancer l'exécution. Voici le nouveau code nécessaire :

```
<%@page import ="org.soa.tp2.GestionService" %>
<%@page import ="org.soa.tp2.Compte" %>
<%@page import ="org.soa.tp2.GestionWS" %>
<%@page import ="jakarta.xml.ws.BindingProvider" %>
<%@page import ="java.util.ArrayList" %>
<%@page import ="java.util.Iterator" %>
<html>
<body>
<h2>Hello World!  Services WEB...</h2>
<%
GestionService stub2 = new GestionWS().getGestionServicePort();

        // Afficher les comptes
        //Afficher une liste de comptes créés par le service web "GestionWS"
        ArrayList<Compte> lc= (ArrayList<Compte>)stub2.getComptes();
        for (Iterator it = lc.iterator(); it.hasNext();)
        {
            Compte c= (Compte)it.next();
            out.println(c.getCode()+":"+c.getSolde()+"<br>");
        }

%>
</body> </html>
```

NB :

- **Ne pas oublier de générer les proxys du service « GestionWS »**
- **Ajouter les deux dépendances « jakarta.xml.ws-api » et « rt » pour corriger les erreurs de compilation (voir Atelier 04)**
- **Ne pas oublier de publier, tout d'abord, le service web « GestionWS » dans le projet « jax-ws-serveur »**

13. Ré exécuter la page « **index.jsp** » et visualiser le résultat :



14. Réaliser les modifications nécessaires dans le code du service web (côté serveur) pour faire fonctionner la nouvelle version du fichier « **index.jsp** » comme suit (consommer de nouvelles fonctionnalités):

```
<%@page import ="org.soa.tp2.GestionService" %>
<%@page import ="org.soa.tp2.Compte" %>
<%@page import ="org.soa.tp2.GestionWS" %>
<%@page import ="jakarta.xml.ws.BindingProvider" %>
<%@page import ="java.util.ArrayList" %>
<%@page import ="java.util.Iterator" %>
<html>
<body>
<h2>Hello World!  Services WEB...</h2>
<%
GestionService stub2 = new GestionWS().getGestionServicePort();

         // Afficher les comptes
         //Afficher une liste de comptes créés par le service web "GestionWS"
         ArrayList<Compte> lc= (ArrayList<Compte>)stub2.getComptes();
         for (Iterator it = lc.iterator(); it.hasNext();)
         {
             Compte c= (Compte)it.next();
             out.println(c.getCode()+":"+c.getSolde()+"<br>");
         }


         // Ajouter un premier compte
         Compte ct1 =new Compte();
         ct1.setCode(1);
         ct1.setSolde(20.2);
         stub2.ajouterCompte(ct1);
         // Ajouter un deuxième compte
         Compte ct2 =new Compte();
         ct2.setCode(2);
         ct2.setSolde(100.0);
         stub2.ajouterCompte(ct2);
         //Afficher la liste des comptes enregistrés
         out.println("----------------------<br>");
         out.println("Comptes enregistrés...."+"<br>");
         ArrayList<Compte> lct= (ArrayList<Compte>)stub2.getListeComptes();
         for (Iterator it = lct.iterator(); it.hasNext();)
         {
             Compte c= (Compte)it.next();
             out.println(c.getCode()+":"+c.getSolde()+"<br>");
         }
         // Supprimer le compte ayant le code 1
         out.println("----------------------"+"<br>");
         out.println("Supprimer le compte ayant le code 1...."+"<br>");
         stub2.supprimerCompte(1);
         //Afficher la liste des comptes enregistrés
         out.println("----------------------"+"<br>");
         out.println("Comptes enregistrés...."+"<br>");
         ArrayList<Compte> lct2= (ArrayList<Compte>)stub2.getListeComptes();
         for (Iterator it = lct2.iterator(); it.hasNext();)
         {
             Compte c= (Compte)it.next();
             out.println(c.getCode()+":"+c.getSolde()+"<br>");
         }

%>
</body> </html>
```

15. Relancer la même application web en utilisant le serveur « Tomcat ».

Indications :

- Ajouter dans le fichier « **pom.xml** » la dépendance de la bibliothèque de servlet à l'intérieur de la balise « **dependencies** »:

```
<!-- Servlet Library -->
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>3.1.0</version>
   <scope>provided</scope>
</dependency>
```

- Ajouter dans le fichier « **pom.xml** » le plugin de Tomcat à l'intérieur de la balise « **plugins** »:

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <!-- Config: contextPath and Port (Default : 8080) -->

  <configuration>
      <path>/</path>
      <port>8899</port>
  </configuration>

</plugin>
```