TP n°4 : SOAP en Java

Objectif

Ce TP sera enseigné en mettant l'accent sur :

- ✓ La création d'un service Web SOAP en Java
- ✓ La récupération de WSDL d'un service SOAP grâce à une requête http
- ✓ Le teste des services Web SOAP avec l'outil SoapUI

Rappel

- Un ***service Web SOAP*** (Simple Object Acces Protocol) permet l'appel d'une méthode d'un objet distant en utilisant un protocole web pour le transport (http, SMTP ou FTP) et XML pour formater les échanges.
- Les ***services Web SOAP*** fonctionnent sur le principe client/serveur :
  - ✓ Un client appelle les services Web,
  - ✓ Un serveur traite la demande et renvoie le résultat au client,
  - ✓ Le client utilise le résultat,
- Les ***services Web SOAP*** peuvent être utilisés pour :
  - ✓ Appeler une méthode d'un service (SOAP RPC)
  - ✓ Échanger un message avec un service (SOAP Messaging)

- Un ***service Web SOAP*** utilise WSDL (acronyme de Web Service Description Language) pour fournir sa description de manière indépendante de tout langage afin de permettre son utilisation.
- Un WSDL fournit cette description dans un document XML. Il contient toutes les informations nécessaires à l'invocation du service qu'il décrit :
  - ✓ Une définition du service,
  - ✓ Les types de données utilisés notamment dans le cas de types complexes,
  - ✓ Les opérations utilisables,
  - ✓ Le protocole utilisé pour le transport
  - ✓ L'adresse d'appel.

Travail demandé

Préparation de votre environnement :

1- Installez « SoapUI » qui est un outil qui permet de tester les fonctions (opérations) offerte par un service Web SOAP.
Il suffit d'exécuter *« SoapUI-x64-5.7.2.exe »*

2- Installez « JDK21 ». ***(Facultative)***
Il suffit d'exécuter *« jdk-21_windows-x64_bin.exe »*

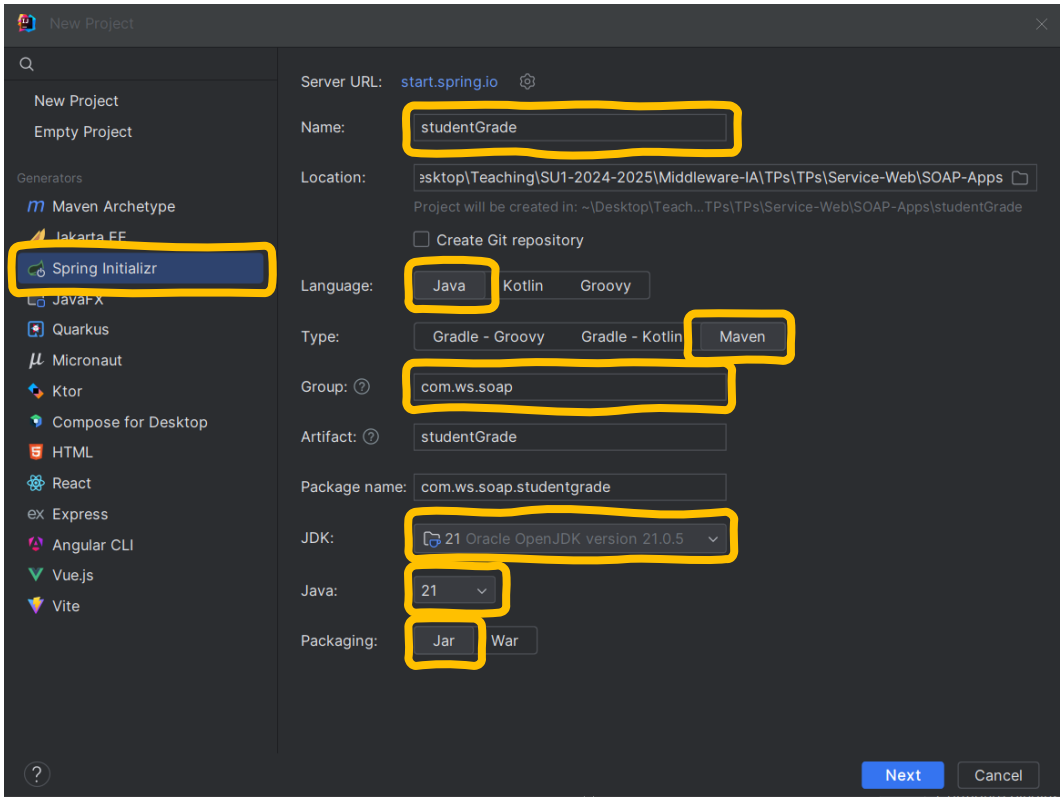
Énoncé

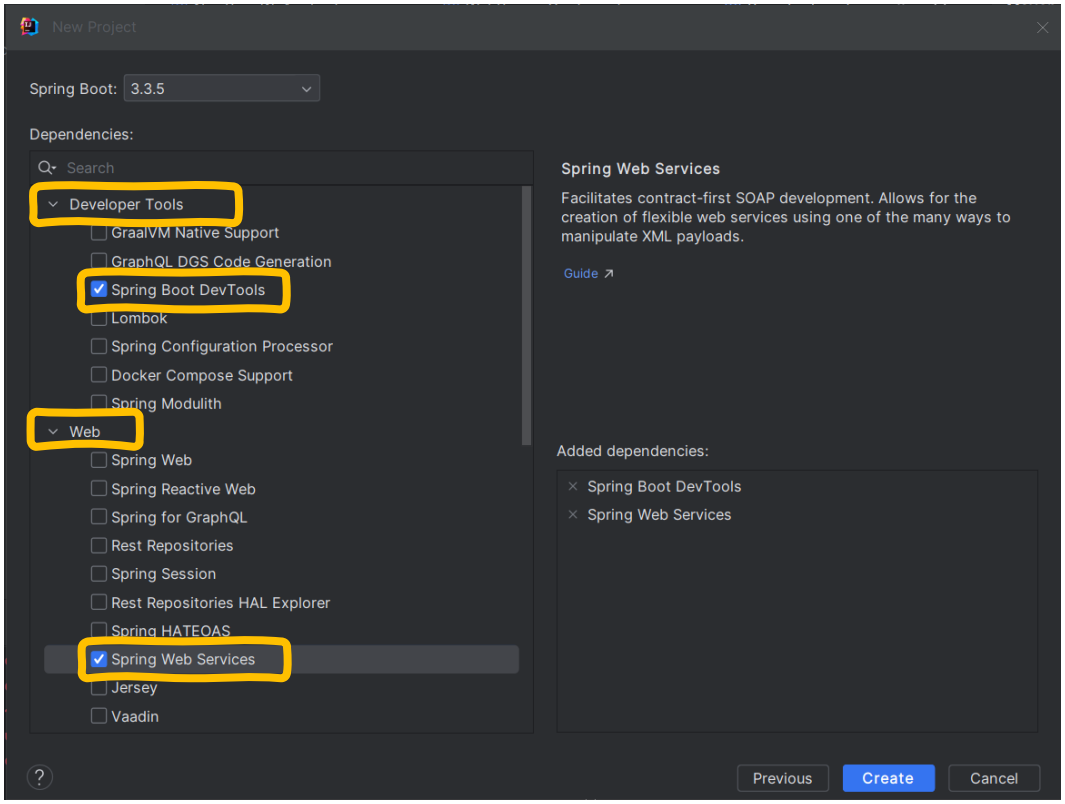
Exercice 1 : Premiers pas avec SOAP

NB : cet exercice est un simple tutoriel permettant de mettre en œuvre un service Web SOAP en Java.

Un exemple simple d'application qui permet d'échanger la note d'un étudiant entre un client et un fournisseur d'un service Web SOAP.

1- Créez un projet Spring Boot basé sur Maven *« StudentGrade »* dans le même espace de travail *« TPs-Middlewares-Prénom-Nom »*,





2- Modifiez le fichier *« pom.xml »* en rajoutant la dépendance *« wsdl4j »* suivante :

```
<dependency>
  <groupId>wsdl4j</groupId>
  <artifactId>wsdl4j</artifactId>
</dependency>
```

Vous devez vous assurer que ces dépendances sont automatiquement prises en compte dans le fichier POM :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-services</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

3- Dans votre fichier *« pom.xml »*, ajoutez le plugin maven *« jaxb2-maven-plugin »* ci-dessous à l'intérieur de la balise <*plugins*> dans <*build*> pour convertir des schémas XML (fichiers XSD) en POJO (Plain Old Java Object)

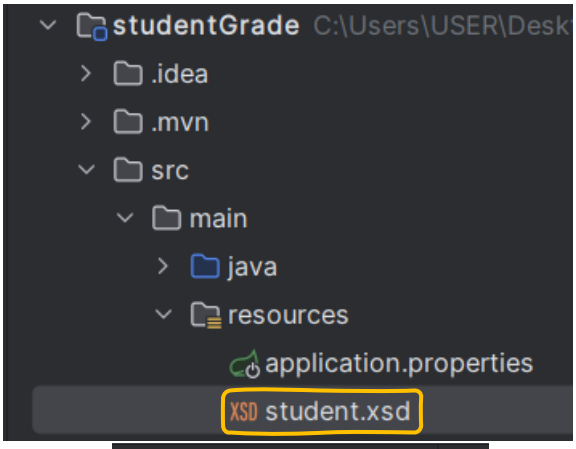
```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxb2-maven-plugin</artifactId>
    <version>3.2.0</version>
    <executions>
        <execution>
            <id>xjc</id>
            <goals>
                <goal>xjc</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <sources>
            <source>src/main/resources/student.xsd</source>
        </sources>
        <outputDirectory>src/main/java</outputDirectory>
        <clearOutputDir>false</clearOutputDir>
    </configuration>
</plugin>
```

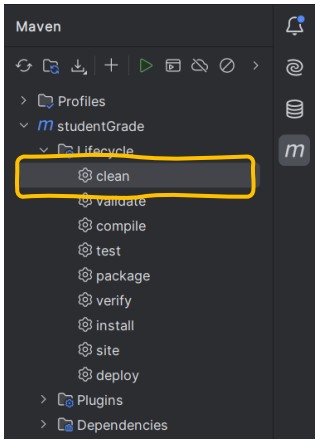
Vous devez mettre à jour le fichier « pom.xml » pour tenir compte de ces modifications.

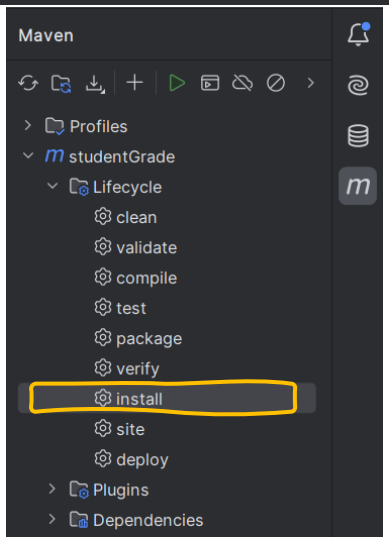
4- Ensuite, créez le fichier XSD *« student.xsd »* dans le dossier *« ressources »* (src->main->resources) pour définir la requête *« getGradeRequest »* et la réponse *« getGradeResponse ».*

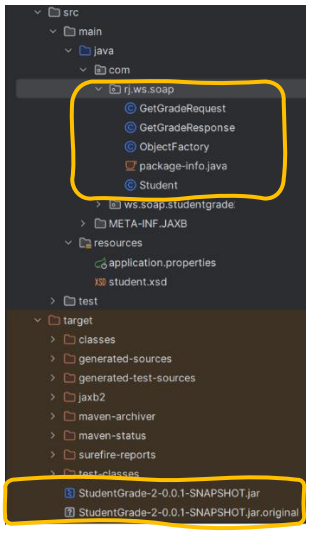
```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://soap.ws.rj.com/"
           targetNamespace="http://soap.ws.rj.com/" elementFormDefault="qualified">

    <xs:element name="getGradeRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getGradeResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student" type="tns:student"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="student">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="grade" type="xs:double"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

5- Maintenant, nettoyez et installez à l'aide de maven pour générer les POJO.









Sous le package *« ws.soap.studentgrade » :*

6- Créez une nouvelle classe « StudentEndpoint » représentant la méthode du service Web :

```
@Endpoint
public class StudentEndpoint {
    no usages
    @PayloadRoot(namespace = "http://soap.ws.rj.com/", localPart = "getGradeRequest")
    @ResponsePayload
    public GetGradeResponse getGrade(@RequestPayload GetGradeRequest request)
    {
        Student student = new Student();
        if (request.getName().equalsIgnoreCase( anotherString: "Anonymous"))
        {
            student.setGrade(10.0);
        }
        else
        {
            student.setGrade(00.0);
        }
        GetGradeResponse response = new GetGradeResponse();
        response.setStudent(student);
        return response;
    }
}
```

7- Ajoutez la classe *« SOAPWebServiceConfig »* pour configurer le service Web et générer le WSDL :

```
@Configuration
@EnableWs
public class SOAPWebServiceConfig extends WsConfigurerAdapter {

    @Bean
    public ServletRegistrationBean<MessageDispatcherServlet> messageDispatcherServlet(ApplicationContext applicationContext)
    {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean<>(servlet, ...urlMappings: "/ws/*");
    }

    @Bean
    public XsdSchema studentSchema()
    {
        return new SimpleXsdSchema(new ClassPathResource("student.xsd"));
    }

    @Bean(name = "student")
    public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema scoreSchema)
    {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("StudentPort");
        wsdl11Definition.setLocationUri("/ws");
        wsdl11Definition.setTargetNamespace("http://soap.ws.rj.com/");
        wsdl11Definition.setSchema(studentSchema());
        return wsdl11Definition;
    }
}
```

8- Démarrez l'application :

Méthode 1 :
Effectuez la compilation de maven à l'aide de « mvn clean install » et démarrez l'application à l'aide de la commande *« java -jar .\target\studentGrade-0.0.1-SNAPSHOT.jar »*.

Ou

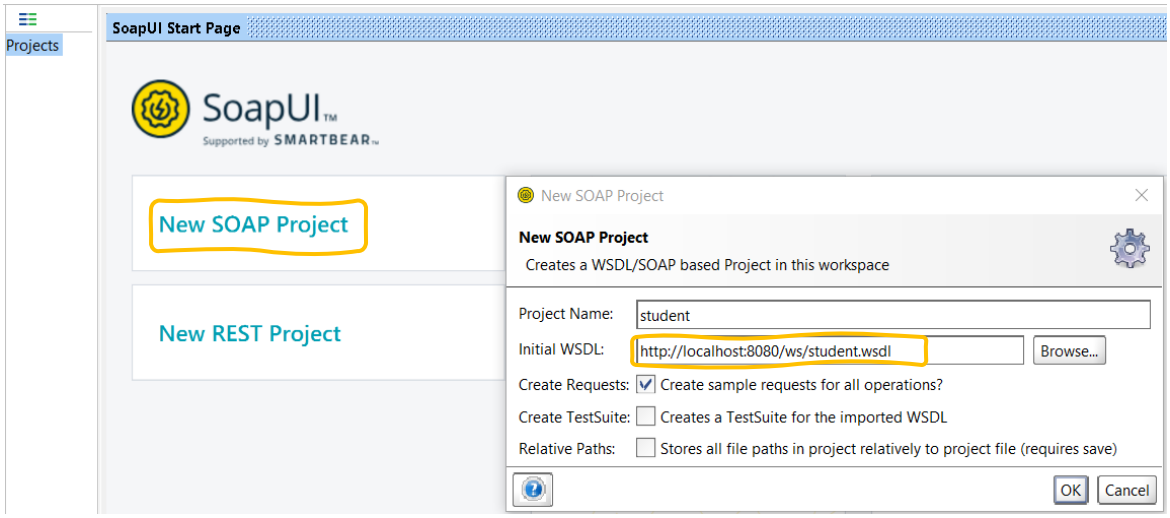
Méthode 2 :
Exécutez tout simplement la classe *« StudentGradeApplication »* en tant qu'application Java.

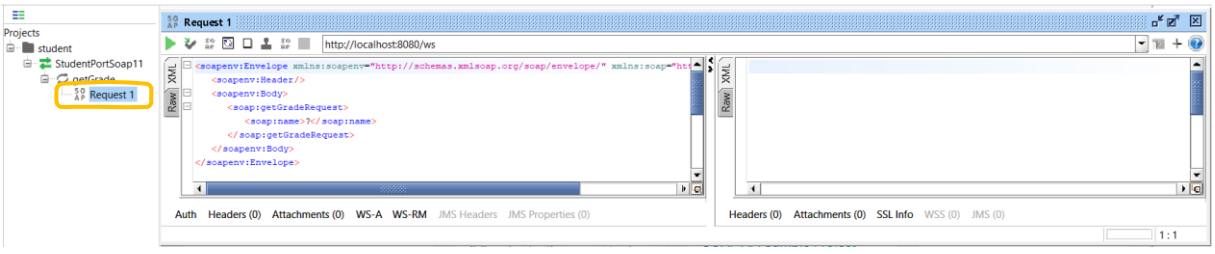
➔ Cela fera apparaître un serveur Tomcat sur le port par défaut 8080 et l'application y sera déployée.

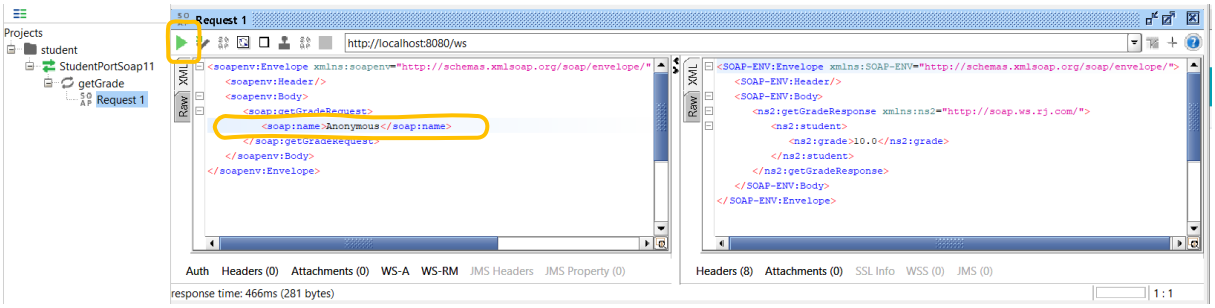
9- Accédez maintenant à *« http://localhost:8080/ws/student.wsdl »* pour voir si le fichier WSDL du service Web s'affiche correctement.

10- Testez le service Web avec le client SoapUI :

a. Lancez SoapUI
b. Créez un nouveau projet SOAP
c. Saisissiez l'adresse du fichier WSDL du service Web dans le champ Initial WSDL
d. Invoquez l'opération de ce service en tapant le nom d'un étudiant « Anonymous » et puis soumettez le message SOAP
e. Vérifiez le résultat de la réponse
f. Réinvoquez l'opération en utilisant votre prénom.







Exercice 2 : Calculator avec SOAP

- Créez un projet Spring Boot basé sur Maven *« Calculator »* qui offre un nouveau service Web SOAP. Ce service définit deux opérations :
    - ***Sum*** : cette opération permet de calculer la somme des nombres entiers. Elle nécessite deux paramètres « a » et « b ».
    - ***Sub*** : cette opération permet de calculer la différence entre des nombres entiers. Elle nécessite deux paramètres « a » et « b ».

- Complétez le bout du fichier XSD *« calculator.xsd »* suivant :

```
<xs:element name="getSumRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="a" type="xs:int"/>
            <xs:element name="b" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="getSumResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="c" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Implémentez la classe « CalculatorEndpoint » représentant les méthodes du service Web.
- Implémentez la classe « SOAPWebServiceConfig » pour configurer le service Web et générer le WSDL.
- Testez le service Web en utilisant SoapUI en faisant plusieurs tests.