Architecture orientée services (SOA) SOAP web service (Spring Boot)

I. Mouakher

Partie I : Producing a SOAP web service (Spring Boot)

Approache pour développer des WS SOAP

- **Contract first**
  - **WSDL → Java**
- Contract Last
  - Java → WSDL

Create Spring Boot Project

Technology Stack

- JDK 1.8, IntelliJ, Maven – Development environment
- Spring-boot – Underlying application framework
- wsdl4j – for publishing WSDL for our Service
- Postman – for testing our service
- JAXB maven plugin – for code generation

Starting with Spring Initializr

- https://start.spring.io/
- Ajouter les dépendences : **Spring Web** et **Spring Web Services.**

Starting with Spring Initializr

spring initializr

Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot
- ( ) 2.6.0 (SNAPSHOT)
- ( ) 2.6.0 (RC1)
- ( ) 2.5.7 (SNAPSHOT)
- (●) 2.5.6
- ( ) 2.4.13 (SNAPSHOT)
- ( ) 2.4.12

Project Metadata

Group	de.tekup
Artifact	producing-web-service
Name	producing-web-service
Description	Demo project for Spring Boot
Package name	de.tekup.country.ws
Packaging	(●) Jar ( ) War
Java	( ) 17 ( ) 11 (●) 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Web Services WEB
Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

Add the Spring-WS dependency

- Modifier le fichier pom.xml comme suit:

```
<!-- tag::springws[] -->
<dependency>
  <groupId>wsdl4j</groupId>
  <artifactId>wsdl4j</artifactId>
</dependency>
<!-- end::springws[] -->
```

Create an XML Schema to Define the Domain

Create an XML Schema to Define the Domain

- L'approche *contract first* nous oblige à créer d'abord le domaine (méthodes et paramètres) pour notre service.
- Nous allons utiliser un fichier de schéma XML (XSD) que Spring WS exportera automatiquement en WSDL.
- Create an XSD file with operations to return a country's name, population, capital, and currency.

Fichier countries.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://www.tekup.de/country/ws/ProducingWebService"
        targetNamespace="http://www.tekup.de/country/ws/ProducingWebService"
        elementFormDefault="qualified">

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

  <xs:element name="getCountryResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="country" type="tns:country"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="country">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="population" type="xs:int"/>
      <xs:element name="capital" type="xs:string"/>
      <xs:element name="currency" type="tns:currency"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="currency">
    <xs:restriction base="xs:string">
      <xs:enumeration value="GBP"/>
      <xs:enumeration value="EUR"/>
      <xs:enumeration value="PLN"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Generate Domain Classes Based on an XML Schema

- Nous allons maintenant générer les classes Java à partir du fichier XSD défini dans la section précédente. Lejaxb2-maven-plugin le fera automatiquement pendant la construction. Le plugin utilise l'outil XJC comme moteur de génération de code. XJC compile le fichier des schéma XSD en classes entièrement Java.

Generate Domain Classes Based on an XML Schema

```
<!-- tag::xsd[] -->
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>2.5.0</version>
  <executions>
    <execution>
      <id>xjc</id>
      <goals>
        <goal>xjc</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <sources>
      <source>${project.basedir}/src/main/resources/countries.xsd</source>
    </sources>
    <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
    <clearOutputDir>false</clearOutputDir>
  </configuration>
</plugin>
<!-- end::xsd[] -->
```

- Par défaut les classe généré seront placées dans le répertoire target/generated-sources/jaxb/.
- Nous ajoutons cette configuration dans Lejaxb2-maven-plugin pour modifier son emplacement (voir diapo précédente)

```
<outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
```

- Clean et install le projet

Generate Domain Classes Based on an XML Schema

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help producing-web-service - pom.xml

Add Configuration...

producing-web-service 〉 pom.xml

Project Files

producing-web-service D:\cours\utk\WebServices\Ines\SOAP\TP
- .idea
- .mvn
- src
  - main
    - java
      - de
        - tekup
          - country
            - ws
              - producingwebservice
                - Country
                - Currency
                - GetCountryRequest
                - GetCountryResponse
                - ObjectFactory
                - package-info.java
                - ProducingWebServiceApplication
    - META-INF
  - resources

ProducingWebServiceApplication.java | countries.xsd | pom.xml (producing-web-service)

```
67                <sources>
68                    <source>${project.basedir}/src/main/resourc
69                </sources>
70                <outputDirectory>${project.basedir}/src/main/ja
71                <clearOutputDir>false</clearOutputDir>
72            </configuration>
73        </plugin>
74    </plugins>
75 </build>
76
77 </project>
78
```

project 〉 build 〉 plugins 〉 plugin

Maven

- producing-web-service
  - Lifecycle
    - clean
    - validate
    - compile
    - test
    - package
    - verify
    - install
    - site
    - deploy
  - Plugins
  - Dependencies

Run: producing-web-service [clean,install]

producing-web-ser 1 min, 11 sec, 681 ms

```
[INFO] Installing D:\cours\utk\WebServices\Ines\SOAP\TP\producing-web-service\pom.xml to C:\Users\meriem\.m2\repository\de\tekup\producing-web-ser
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:00 min
[INFO] Finished at: 2021-11-14T11:55:11+01:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

Create Country Repository

- Afin de fournir des données au service Web, créez un country repository. Vous créez une implémentation un dummy country repository avec des données codées en dur. Ci-dessous le listing:

- n order to provide data to the web service, create a country repository. In this guide, you create a dummy country repository implementation with hardcoded data. The following listing (from src/main/java/com/example/producingwebservice/CountryRepository.java) shows how to do so:

```java
package de.tekup.country.ws;

import de.tekup.country.ws.producingwebservice.Country;
import de.tekup.country.ws.producingwebservice.Currency;
import org.springframework.stereotype.Component;
import org.springframework.util.Assert;

import javax.annotation.PostConstruct;
import java.util.HashMap;
import java.util.Map;
@Component
public class CountryRepository {
    private static final Map<String, Country> countries = new HashMap<>();


    @PostConstruct
    public void initData() {
        Country spain = new Country();
        spain.setName("Spain");
        spain.setCapital("Madrid");
        spain.setCurrency(Currency.EUR);
        spain.setPopulation(46704314);

        countries.put(spain.getName(), spain);

        Country poland = new Country();
        poland.setName("Poland");
        poland.setCapital("Warsaw");
        poland.setCurrency(Currency.PLN);
        poland.setPopulation(38186860);

        countries.put(poland.getName(), poland);

        Country uk = new Country();
        uk.setName("United Kingdom");
        uk.setCapital("London");
        uk.setCurrency(Currency.GBP);
        uk.setPopulation(63705000);

        countries.put(uk.getName(), uk);
    }
    public Country findCountry(String name) {
        Assert.notNull(name, "The country's name must not be null");
        return countries.get(name);
    }
}
```

Implementing the SOAP Endpoint

- To create a service endpoint, you need only a POJO with a few Spring WS annotations to handle the incoming SOAP requests.

Create Country Service Endpoint

```
package de.tekup.country.ws;

import de.tekup.country.ws.producingwebservice.GetCountryRequest;
import de.tekup.country.ws.producingwebservice.GetCountryResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

@Endpoint
public class CountryEndpoint {

    private static final String NAMESPACE_URI = "http://www.tekup.de/country/ws/ProducingWebService";

    private CountryRepository countryRepository;

    @Autowired
    public CountryEndpoint(CountryRepository countryRepository) {
        this.countryRepository = countryRepository;
    }

    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getCountryRequest")

    @ResponsePayload
    public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request) {
        GetCountryResponse response = new GetCountryResponse();
        response.setCountry(countryRepository.findCountry(request.getName()));

        return response;
    }

}
```

Create Country Service Endpoint

- The @Endpoint annotation registers the class with Spring WS as a potential candidate for processing incoming SOAP messages.
- The @PayloadRoot annotation is then used by Spring WS to pick the handler method, based on the message's namespace and localPart.
- The @RequestPayload annotation indicates that the incoming message will be mapped to the method's request parameter.
- The @ResponsePayload annotation makes Spring WS map the returned value to the response payload.

Configure Web Service Beans

Configure Web Service Beans

- Spring WS uses a different servlet type for handling SOAP messages: MessageDispatcherServlet. It is important to inject and set ApplicationContext to MessageDispatcherServlet. Without that, Spring WS will not automatically detect Spring beans.
- Naming this bean messageDispatcherServlet does not replace Spring Boot's default DispatcherServlet bean.
- DefaultMethodEndpointAdapter configures the annotation-driven Spring WS programming model. This makes it possible to use the various annotations, such as @Endpoint (mentioned earlier).
- DefaultWsdl11Definition exposes a standard WSDL 1.1 by using XsdSchema

```java
package de.tekup.country.ws;

import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurerAdapter;
import org.springframework.ws.transport.http.MessageDispatcherServlet;
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.xml.xsd.SimpleXsdSchema;
import org.springframework.xml.xsd.XsdSchema;

@EnableWs
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {

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

    }

    @Bean(name = "countries")
    public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema countriesSchema) {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("CountriesPort");
        wsdl11Definition.setLocationUri("/ws");
        wsdl11Definition.setTargetNamespace("http://www.tekup.de/country/ws/ProducingWebService");
        wsdl11Definition.setSchema(countriesSchema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema countriesSchema() {
        return new SimpleXsdSchema(new ClassPathResource("countries.xsd"));
    }
}
```

Test

Consulter le webservice

- http://localhost:8080/ws/countries.wsdl

localhost:8080/ws/countries.wsdl

Apps YouTube Maps Gmail

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


```
▼<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch="http://www.tekup.de/country/ws/ProducingWebService" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.tekup.de/country/ws/ProducingWebService" targetNamespace="http://www.tekup.de/country/ws/ProducingWebService">
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://www.tekup.de/country/ws/ProducingWebService">
      ▼<xs:element name="getCountryRequest">
        ▼<xs:complexType>
          ▼<xs:sequence>
              <xs:element name="name" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      ▼<xs:element name="getCountryResponse">
        ▼<xs:complexType>
          ▼<xs:sequence>
              <xs:element name="country" type="tns:country"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      ▼<xs:complexType name="country">
        ▼<xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="population" type="xs:int"/>
            <xs:element name="capital" type="xs:string"/>
            <xs:element name="currency" type="tns:currency"/>
          </xs:sequence>
        </xs:complexType>
      ▼<xs:simpleType name="currency">
        ▼<xs:restriction base="xs:string">
            <xs:enumeration value="GBP"/>
            <xs:enumeration value="EUR"/>
            <xs:enumeration value="PLN"/>
          </xs:restriction>
        </xs:simpleType>
```

Tester avec Posman

1. Open a new request tab in Postman and enter your SOAP endpoint URL in the address field.
    - http://localhost:8080/ws/countries.wsdl
2. Select **POST** from the request method dropdown list.
3. In the **Body** tab, select **raw** and choose **XML** from the dropdown list.
4. Enter your XML in the text entry area
5. Open the request Headers. Deselect the Content-Type header Postman added automatically. Add a new row with Content-Type in the Key field and text/xml in the Value field.

(From https://learning.postman.com/docs/sending-requests/supported-api-frameworks/making-soap-requests/)

Tester avec Posman

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:prod="http://www.tekup.de/country/ws/ProducingWebService">
  <soapenv:Header/>
  <soapenv:Body>
    <prod:getCountryRequest>
      <prod:name>Spain</prod:name>
    </prod:getCountryRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Tester avec Posman

POST http://localhost:8...

No Environment

http://localhost:8080/ws/countries.wsdl

Save

POST http://localhost:8080/ws/countries.wsdl Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL XML Beautify

```
3   <soapenv:Header/>
4   <soapenv:Body>
5     <prod:getCountryRequest>
6       <prod:name>Spain</prod:name>
7     </prod:getCountryRequest>
8   </soapenv:Body>
9 </soapenv:Envelope>
10
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 17 ms Size: 669 B Save Response

Pretty Raw Preview Visualize XML

```
1  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
2      <SOAP-ENV:Header/>
3      <SOAP-ENV:Body>
4          <ns2:getCountryResponse xmlns:ns2="http://www.tekup.de/country/ws/ProducingWebService">
5              <ns2:country>
6                  <ns2:name>Spain</ns2:name>
7                  <ns2:population>46704314</ns2:population>
8                  <ns2:capital>Madrid</ns2:capital>
9                  <ns2:currency>EUR</ns2:currency>
10             </ns2:country>
11         </ns2:getCountryResponse>
12     </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

Partie II : Consuming a SOAP web service (Spring Boot)

Create Spring Boot Project

spring initializr

Project

- (●) Maven Project
- ( ) Gradle Project

Language

- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot

- ( ) 2.6.0 (SNAPSHOT)
- ( ) 2.6.0 (RC1)
- ( ) 2.5.7 (SNAPSHOT)
- (●) 2.5.6
- ( ) 2.4.13 (SNAPSHOT)
- ( ) 2.4.12

Project Metadata

Group	de.tekup
Artifact	consuming-web-service
Name	consuming-web-service
Description	consuming web service
Package name	de.tekup.country.ws.consume
Packaging	(●) Jar ( ) War
Java	( ) 17 ( ) 11 (●) 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Web Services WEB

Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.

Changer pom.xml

```
<!-- tag::dependency[] -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web-services</artifactId>


 <exclusions>
     <exclusion>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-tomcat</artifactId>
     </exclusion>
  </exclusions>



</dependency>
<!-- end::dependency[] -->
```

Changer pom.xml

```xml
<!-- tag::profile[] -->
<profiles>
  <profile>
    <id>java11</id>
    <activation>
      <jdk>[11,)</jdk>
    </activation>

    <dependencies>
      <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
      </dependency>
    </dependencies>
  </profile>
</profiles>
<!-- end::profile[] -->
```

Generate Domain Objects Based on a WSDL

Generate Domain Objects Based on a WSDL

- The interface to a SOAP web service is captured in WSDL. JAXB provides a way to generate Java classes from WSDL. You can find the WSDL for the country service at http://localhost:8080/ws/countries.wsdl.

- To generate Java classes from the WSDL in Maven, you need the following plugin setup:

Changer pom.xml

```xml
<!-- tag::wsdl[] -->
<plugin>
  <groupId>org.jvnet.jaxb2.maven2</groupId>
  <artifactId>maven-jaxb2-plugin</artifactId>
  <version>0.14.0</version>
  <executions>
    <execution>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <schemaLanguage>WSDL</schemaLanguage>
    <generatePackage>de.tekup.country.ws.consume.countryws</generatePackage>
    <generateDirectory>${project.basedir}/src/main/java</generateDirectory>
    <schemas>
      <schema>
        <url>http://localhost:8080/ws/countries.wsdl</url>
      </schema>
    </schemas>
  </configuration>

</plugin>
<!-- end::wsdl[] -->
```

Generate classes for the WSDL

- This setup will generate classes for the WSDL found at the specified URL, putting those classes in the com.example.consumingwebservice.wsdl package. To generate that code run ./mvnw compile.

Generate classes for the WSDL

consuming-web-service > src

Project

- consuming-web-service D:\cours\utk\WebServices\Ines\SOAP\
  - .idea
  - .mvn
  - src
    - main
      - java
        - de.tekup.country.ws.consume
          - countryws
            - Country
            - Currency
            - GetCountryRequest
            - GetCountryResponse
            - ObjectFactory
            - package-info.java
          - ConsumingWebServiceApplication
        - META-INF
      - resources
    - test
  - target
  - .gitignore

pom.xml (consuming-web-service)

```
<groupId>org.jvnet.jaxb2.maven2</groupId>
<artifactId>maven-jaxb2-plugin</artifactId>
<version>0.14.0</version>
<executions>
    <execution>
        <goals>
            <goal>generate</goal>
        </goals>
    </execution>
</executions>
<configuration>
    <schemaLanguage>WSDL</schemaLanguage>
    <generatePackage>de.tekup.country.ws.consume.c
    <generateDirectory>${project.basedir}/src/main/
    <schemas>
        <schema>
            <url>http://localhost:8080/ws/countrie
        </schema>
```

project > build > plugins > plugin > configuration > generatePackage

Maven

- consuming-web-service
  - Lifecycle
    - clean
    - validate
    - compile
    - test
    - package
    - verify
    - install
    - site
    - deploy
  - Plugins
  - Dependencies

Run: consuming-web-service [compile]

consuming-web-service 36 sec, 371 ms

```
[INFO] Compiling 7 source files to D:\cours\utk\WebServices\Ines\SOAP\TP\consuming-web-service\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  30.507 s
[INFO] Finished at: 2021-11-14T21:11:21+01:00
[INFO] ------------------------------------------------------------------------
```

Create a Country Service Client

Create a Country Service Client

- To create a web service client, you have to extend the WebServiceGatewaySupport class and code your operations, as the following example (from src/main/java/com/example/consumingwebservice/CountryClient.java) shows:

Create a Country Service Client

```
public class CountryClient extends WebServiceGatewaySupport {
    private static final Logger log = LoggerFactory.getLogger(CountryClient.class);

    public GetCountryResponse getCountry(String country) {

        GetCountryRequest request = new GetCountryRequest();
        request.setName(country);

        log.info("Requesting location for " + country);

        GetCountryResponse response = (GetCountryResponse) getWebServiceTemplate()
                .marshalSendAndReceive("http://localhost:8080/ws/countries", request,
                        new SoapActionCallback(
                                "http://www.tekup.de/country/ws/ProducingWebService/GetCountryRequest"));

        return response;
    }

}
```

- The client contains one method (getCountry) that does the actual SOAP exchange.
- In this method, both the GetCountryRequest and the GetCountryResponse classes are derived from the WSDL and were generated in the JAXB generation process (described in Generate Domain Objects Based on a WSDL). It creates the GetCountryRequest request object and sets it up with the country parameter (the name of the country). After printing out the country name, it uses the WebServiceTemplate supplied by the WebServiceGatewaySupport base class to do the actual SOAP exchange. It passes the GetCountryRequest request object (as well as a SoapActionCallback to pass on a SOAPAction header with the request) as the WSDL described that it needed this header in the <soap:operation/> elements. It casts the response into a GetCountryResponse object, which is then returned.

Configuring Web Service Components

Configuring Web Service Components

- Spring WS uses Spring Framework's OXM module, which has the Jaxb2Marshaller to serialize and deserialize XML requests.
- The marshaller is pointed at the collection of generated domain objects and will use them to both serialize and deserialize between XML and POJOs.
- The countryClient is created and configured with the URI of the country service shown earlier. It is also configured to use the JAXB marshaller.

Class CountryConfiguration

```
@Configuration
public class CountryConfiguration {
    @Bean
    public Jaxb2Marshaller marshaller() {
        Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
        // this package must match the package in the <generatePackage> specified in
        // pom.xml
        marshaller.setContextPath("de.tekup.country.ws.consume.countryws");
        return marshaller;
    }

    @Bean
    public CountryClient countryClient(Jaxb2Marshaller marshaller) {
        CountryClient client = new CountryClient();
        client.setDefaultUri("http://localhost:8080/ws");
        client.setMarshaller(marshaller);
        client.setUnmarshaller(marshaller);
        return client;
    }
}
```

Run the Application

```java
@SpringBootApplication
public class ConsumingWebServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(ConsumingWebServiceApplication.class, args);
  }

  @Bean
  CommandLineRunner lookup(CountryClient quoteClient) {
    return args -> {
      String country = "Spain";

      if (args.length > 0) {
        country = args[0];
      }
      GetCountryResponse response = quoteClient.getCountry(country);
      System.err.println(response.getCountry().getCurrency());
    };
  }

}
```

Run the Application

- This application is packaged up to run from the console and retrieve the data for a given country name.

- The main() method defers to the SpringApplication helper class, providing CountryConfiguration.class as an argument to its run() method. This tells Spring to read the annotation metadata from CountryConfiguration and to manage it as a component in the Spring application context.

This application is hard-coded to look up 'Spain'.

Build an executable JAR

- You can build the JAR file with ./mvnw clean package and then run the JAR file.
- You can run the application from the command line:
  - java -jar target/consuming-web-service-0.0.1-SNAPSHOT.jar Spain
  - java -jar target/consuming-web-service-0.0.1-SNAPSHOT.jar Poland
  - ….

Références

- **Producing a SOAP web service**

https://spring.io/guides/gs/producing-web-service/#initial

- **Consuming a SOAP web service**

https://spring.io/guides/gs/consuming-web-service/#initial

- IntelliJ IDE : https://www.jetbrains.com/help/idea/new-project-wizard.html#javafx
- Postman : https://learning.postman.com/docs/sending-requests/supported-api-frameworks/making-soap-requests/