Apache CXF

Apache CXF jest frameworkiem o otwartym źródle kodu, zaprojektowanym do tworzenia serwisów internetowych (ang. *open-source services framework*). Framework ten wspiera wiele starndardów, włacznie z: SOAP, WSI Basic Profile, WSDL, WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security, WS-SecurityPolicy, and WS-SecureConversation.

Framework umożliwia dwa realizację dwóch podejść do implementacji serwisów:

a) od dołu, gdy zaczyna się od implementacji klas Java metody których mają stać się metodami serwisu (po angielsku takie podejście nazywa się: *bottom-up* lub *contract-last*).
b) Od góry, gdy zaczyna się od opisu interfejsów serwisów w języku WSDL (po angielsku takie podejście nazywane jest: *top-down* lub *contract-first*).

Apache CXF implementuje JAX-WS, czyli Java API pozwalające na komunikację z serwisami internetowymi z wykorzystaniem XML (obecnie *Jakarta XML Web Services*, kiedyś *Java API for XML Web Services*). API to przechodziło modyfikacje: JAX-WS 2.0/2.1/2.2 (zobacz też JSR 224: JavaTM API for XML-Based Web Services (JAX-WS) 2.0)

Ponadto Apache CXF implementuje JAX-RS, czyli Java API for RESTful Web Services zdefiniowane w Java EE (zobacz też JSR 370: JavaTM API for RESTful Web Services (JAX-RS 2.1) Specification).

Z uwagi na usunięcie komponentów Java EE z JDK11 pisanie programów bazujących na Apache CXF wymaga obecnie dołączenia do projektów odpowiednich zależności.

Wyjaśniając to na przykładzie: kiedyś, aby wygenerować klasy Java z opisu WSDL wystarczyło użyć `wsimport` – tj. użyć narzędzie znajdujące się w dystrybucji JDK. Począwszy od JDK11 tego się już zrobić nie da. Aby wygenerować klasy Java zwykle korzysta się z pluginu `jaxws-maven-plugin` oraz zależności: `jakarta.xml.ws-api`, `jaxws-rt` and `jaxws-ri`.

Niezłym tutorial o tworzeniu serwisów zamieszczono na stronie:
https://www.baeldung.com/jax-ws

Spring-WS

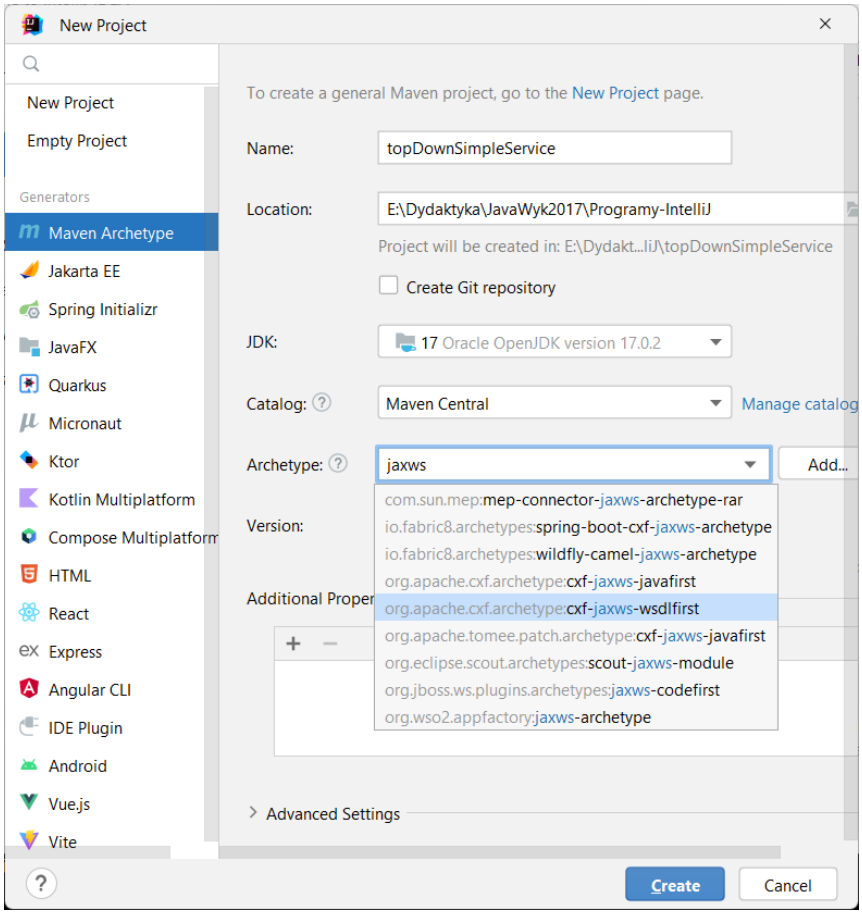
Spring Web Service (Spring-WS) to framework zorientowan na tworzenie serwisów webowych przetwarzających dokumenty (ang. *document-driven Web services*). Z założenia wspierać on ma tworzenie serwisów komunikujących się protokołem SOAP od dołu (bootom-up, contract-first). Pozwalać ma on na tworzenia tworzenie I zarządzanie serwisami, z możliwością przetwarzaniem ładunku niesionego w dokumentach XML na wiele sposobów.

Spring WS wspiera część standardów obsługiwanych przez Apache CXF: SOAP, WS-Security, and WS-Addressing standards.

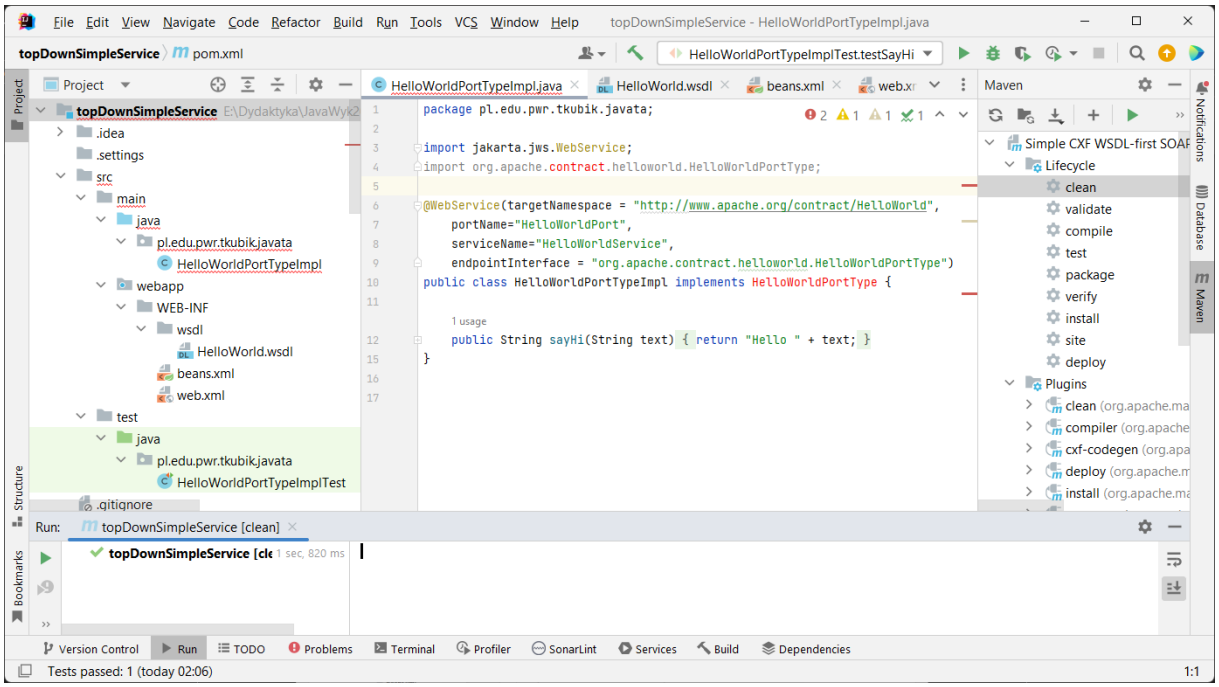
Proszę nie mylić Apache CXF z Spring-WS – to dwa różne frameworki, które implementują JAX-WS.

Projekt mavenowy w IntelliJ serwisu zbudowanego z wykorzystaniem Apache CXF wg podejścia top-down

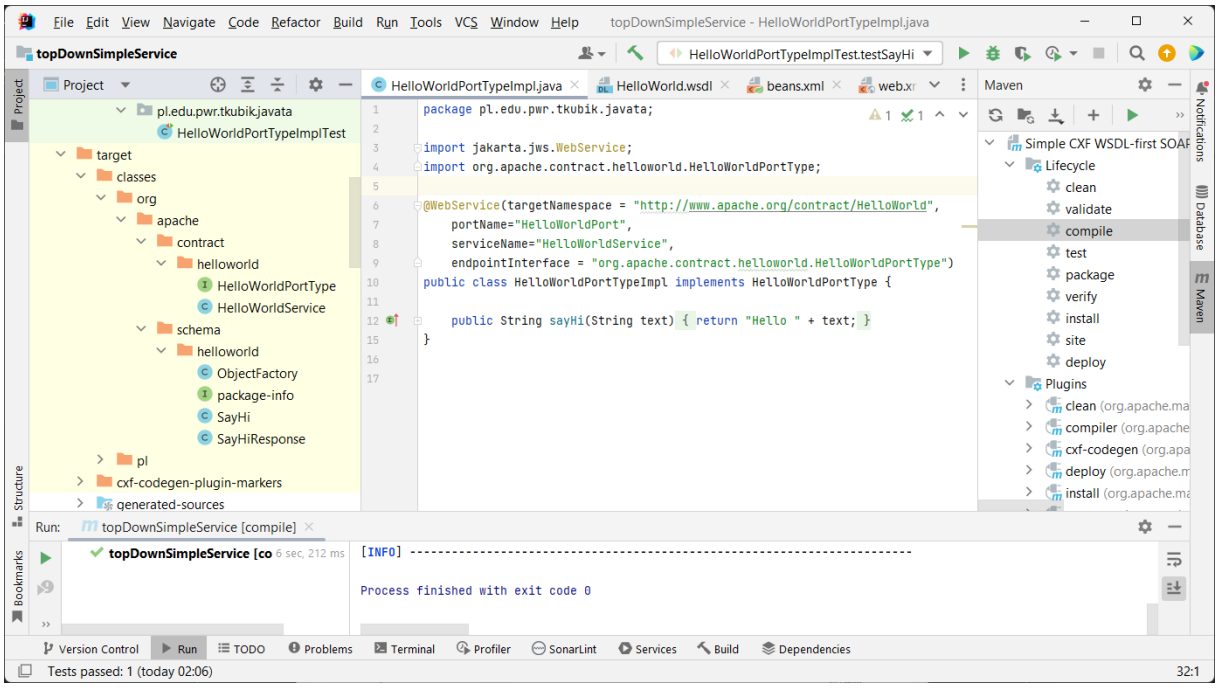
Podstawowy scenariusz tworzenia projektów mavenowych polega na skorzystaniu z jakiegoś archetypu w celu wygenerowania struktury projektu. W omawianym przypadku stosuje się archetyp org.apache.cxf.archetype:cxf-javaws-wsdlfirst.



W wygenerowanym projekcie pojawią się źródła przykładowego serwisu HelloWorld oraz pliki konfiguracyjne włącznie z plikiem HelloWorld.wsdl). Jak się okaże, w niektórych miejscach kod będzie podświetlony na czerwono z powodu braku klas odpowiadających typom zdefiniowanym w pliku WSDL. Aby je wygenerować wystarczy uruchomić mvn compile.



Po tej czynności w katalogu target pojawią się brakujące klasy (za ich pojawienie się odpowiada plugin). Klasy te można byłoby przenieść do katalogu src/main/java. Nie jest to jednak konieczne.



Teraz można zacząć działać dalej – modyfikując projekt do swoich potrzeb. Zamiast dostarczonego HelloWorld.wsdl można podstawić plik z opisem interfesu własnej usługi sieciowej. Jednak po takiej zmianie trzeba będzie skorygować zawartość pliku beans.xml (by wpisy w nim zamieszczone odpowiadały implementacji serwisu).

Jak się okaże, trzeba będzie dokonać kosmetycznej zmiany w pliku web.xml, polegającej na zamianie miejscami dwóch linijek z węzłami display-name oraz servlet-name, by przyjęły postać:

```
<display-name>CXF Servlet</display-name>
<servlet-name>CXFServlet</servlet-name>
```

Ale to jeszcze nie wszystko. Okazuje się, że trzeba brakować będzie zależności w pliku pom.xml.

W sumie trudno z góry określić, jakie to mają być zależności. Ich brak objawia się bowiem podczas próby wdrożenia aplikacji na serwerze aplikacji, jak również podczas wysyłania żądań do serwisu. W tych chwilach mogą pojawić się błędy nieznalezienia jakiejś klasy czy też błędy rzutowania typów. Te błędy to skutek zastosowanych wzorców projektowych (klasy ładowane są dynamicznie) jak również wykorzystanych środowisk uruchomieniowych – JDK oraz Apache Tomcat.

Poniżej pokazano zawartość pliku pom.xml dla konfiguracji środowiska: JDK17 + Apache Tomcat 10. Proszę zwrócić uwagę na pakiety jakarta (których nie było w pom.xml bezpośrednio po wygenerowaniu go z archetypu).

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>pl.edu.pwr.tkubik.javata</groupId>
    <artifactId>topDownSimpleService</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>Simple CXF WSDL-first SOAP project using Spring configuration</name>
    <description>Simple CXF WSDL-first SOAP project using Spring configuration</description>
    <properties>
        <test.server.port>8080</test.server.port>
    </properties>
    <dependencies>
        <dependency>
            <groupId>jakarta.xml.ws</groupId>
            <artifactId>jakarta.xml.ws-api</artifactId>
            <version>4.0.0</version>
        </dependency>
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>6.0.0</version>
        </dependency>
<!--         <dependency>-->
<!--             <groupId>jakarta.xml.bind</groupId>-->
<!--             <artifactId>jakarta.xml.bind-api</artifactId>-->
<!--             <version>4.0.0</version>-->
<!--         </dependency>-->
        <dependency>
            <groupId>com.sun.xml.bind</groupId>
            <artifactId>jaxb-impl</artifactId>
            <version>4.0.0</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>6.0.8</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>6.0.8</version>
        </dependency>
```

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-codegen-plugin</artifactId>
                <version>4.0.1</version>
                <executions>
                    <execution>
                        <configuration>
                            <sourceRoot>
                                target/generated-sources
                            </sourceRoot>
                            <wsdlOptions>
                                <wsdlOption>
                                    <wsdl>
                                       src/main/webapp/WEB-INF/wsdl/HelloWorld.wsdl
                                    </wsdl>
                                    <wsdlLocation>classpath:HelloWorld.wsdl</wsdlLocation>
                                </wsdlOption>
                            </wsdlOptions>
                        </configuration>
                        <goals>
                            <goal>wsdl2java</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>3.3.1</version>
                </plugin>
                <plugin>
                    <!-- mvn clean install tomcat7:run to deploy
                    Look for "Running war on http://xxx" and
                    "Setting the server's publish address to be /yyy"
                    in console output; WSDL browser address will be
                    concatenation of the two: http://xxx/yyy?wsdl
                    -->
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <version>2.2</version>
                    <configuration>
                        <server>TomcatServer</server>
                        <path>/HelloWorld</path>
                    </configuration>
                    <executions>
                        <execution>
                            <id>start-tomcat</id>
                            <goals>
                                <goal>run-war</goal>
                            </goals>
                            <phase>pre-integration-test</phase>
                            <configuration>
                                <port>${test.server.port}</port>
                                <path>/HelloWorld</path>
                                <fork>true</fork>

<useSeparateTomcatClassLoader>true</useSeparateTomcatClassLoader>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
```

```
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <configuration>
                    <projectNameTemplate>[artifactId]-[version]</projectNameTemplate>
                    <wtpmanifest>true</wtpmanifest>
                    <wtpapplicationxml>true</wtpapplicationxml>
                    <wtpversion>2.0</wtpversion>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>

</project>
```

Po zakończeniu implementacji można uruchomić serwis na serwerze aplikacji.

Na początek należy odpowiednio skonfigurować Apache Tomcat oraz mavena (role i użytkownicy w pliku users.xml, informacja o serwisie w pliku settings.xml).

Działa to tak: najpierw trzeba skonfigurować i uruchomić samemu tomcata (wersja 10 lub wyżej, dla wersji wcześniejszych pojawiały się błędy z ładowaniem odpowiednich klas). W konfiguracji użytkowników (`conf\tomcat-users.xml`) trzeba ustawić uprawnienia do administrowania (oczywiście hasło jest tu przykładowe)

```
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<role rolename="manager"/>
    <role rolename="manager-script"/>
<user username="admin" password="pass" roles="manager-gui,admin-gui,manager,manager-script"/>
```

W pliku `pom.xml` znaleźć się ma węzeł

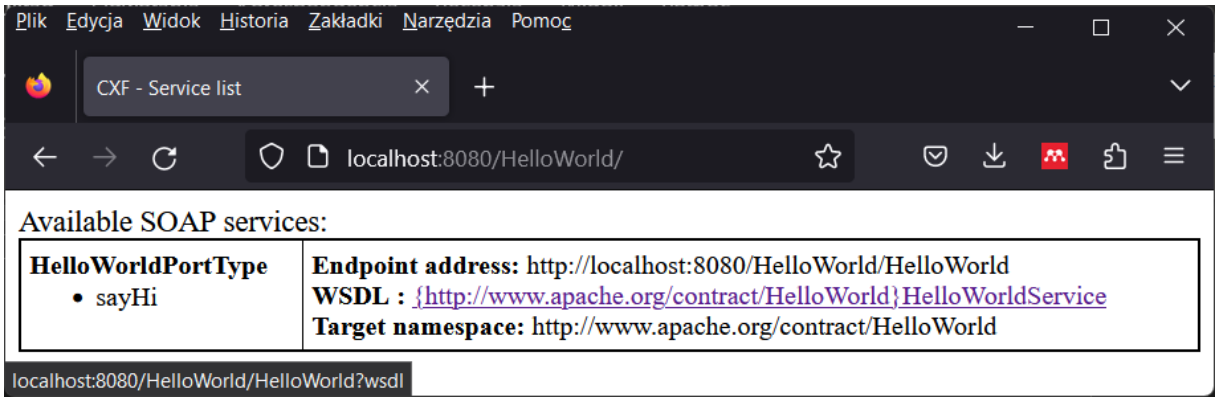
```
<server>TomcatServer</server>
```

Poprzez ten węzeł plugin mavena dowie się, w jakiej sekcji pliku `~.m2\settings.xml` szukać ma parametrów uwierzytelniających do tomcata. W tam właśnie pliku powinien znaleźć się wpis:

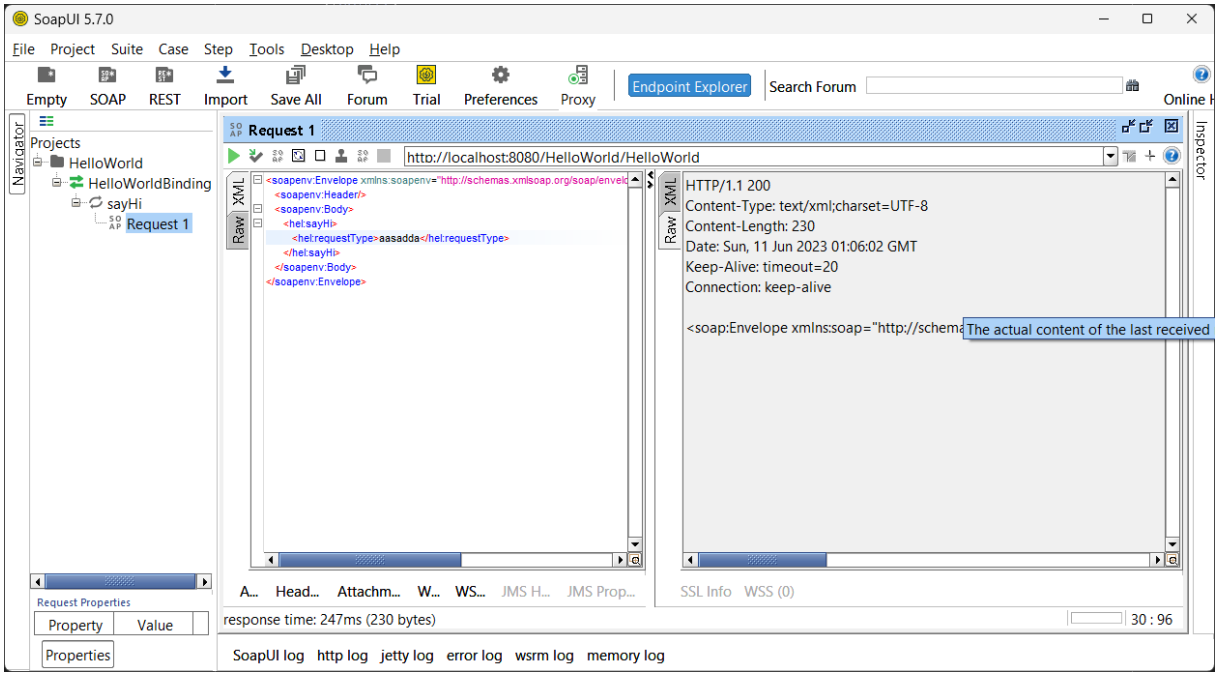
```
    <server>
       <id>TomcatServer</id>
       <username>admin</username>
       <password>pass</password>
    </server>
    <server>
```

Następnie należy odpalić komendę: mvn tomcat7/redeploy

Po jej uruchomieniu pod adresem http://localhost:8080/HelloWorld pojawi się strona z infomacją o serwisach (wynik działania servletu CXF).



Pod wyróżnionym linkiem – http://localhost:8080/HelloWorld?wsdl opublikowany zostanie plik wsdl. Link ten można użyć w projekcie SoapUI – narzędzia, które dostarcza generycznego klienta do serwisów komunikujących się protokołem SOAP (klient jest generowany na podstawie opisu WSDL).



Do budowy serwisów z wykorzystaniem Apache CXF można również wykorzystać inny plugin: jaxws-maven-plugin.

Sposób skonfigurowania tego pluginu opisano na stronie: https://www.baeldung.com/maven-wsdl-stubs

Generalnie jego użycie polega na odpaleniu narzędzia wsimport, które kiedyś było dystrybuowane w JDK, a obecnie jest częścią pakietów Apache CXF.

Poniżej pokazano przykład pliku pom.xml ze skonfigurowanym celem: wsimport (zdefiniowano w nim położenie plików wsdl i generowanego kodu).

Wygenerowany kod nie będzie zawierał źródeł serwisu - taki serwis należy samemu sobie napisać, implementując wygenerowany interfejs. Oczywiście struktura projektu też musi być odpowiednia (by

WEB-INF i inne rzeczy trafiały do generowanego pliku war) oraz dodane powinny być odpowiednie zależności (jeśli serwis miałby zostać wdrożony na serwerze aplikacji).

```
<project
        xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>pl.edu.pwr.tkubik.javata</groupId>
        <artifactId>top-down-jaxws</artifactId>
        <version>1.0-SNAPSHOT</version>
        <name>Archetype - top-down-jaxws</name>
        <url>http://maven.apache.org</url>
        <dependencies>
                <dependency>
                        <groupId>jakarta.xml.ws</groupId>
                        <artifactId>jakarta.xml.ws-api</artifactId>
                        <version>4.0.0</version>
                </dependency>
                <dependency>
                        <groupId>com.sun.xml.ws</groupId>
                        <artifactId>jaxws-rt</artifactId>
                        <version>4.0.0</version>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>com.sun.xml.ws</groupId>
                        <artifactId>jaxws-ri</artifactId>
                        <version>4.0.0</version>
                        <type>pom</type>
                </dependency>
        </dependencies>
        <build>
                <plugins>
                        <plugin>
                                <groupId>com.sun.xml.ws</groupId>
                                <artifactId>jaxws-maven-plugin</artifactId>
                                <version>4.0.0</version>
                                <executions>
                                        <execution>
                                                <goals>
                                                        <goal>wsimport</goal>
                                                </goals>
                                        </execution>
                                </executions>
                                <configuration>

        <wsdlDirectory>${project.basedir}/src/main/resources/</wsdlDirectory>

        <packageName>pl.edu.pwr.tkubik.javata.service</packageName>
        <sourceDestDir>${project.build.directory}/generated-sources/</sourceDestDir>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```