Projekt mavenowy wykorzystujący Apache CXF (w eclipse JEE)

Apache CXF

Framework (dokumentacja na stronie: https://cxf.apache.org/docs/index.html) pozwala na:

- generowanie opisu WSDL z klas Javy (bottom-up) oraz klas Javy z opisu WSDL
- tworzenie serwerowych endpointów przyjmujących żądania (provider API)
- tworzenie klientów wysyłających xml-owe wiadomości do serwerowych endpointów (dispatch API)

Framework wspiera:

- składnię Spring 2.0 XML, co ułatwia deklarowanie edpointów „wypiekanych" przez Spring oraz wstrzykiwanie klientów do kodu aplikacji.
- dowiązanie HTTP poprzez stosowanie adnotacji (@Get, @HttpResource,...), co ułatwia tworzenie serwisów restowych
- standardy WS-*: WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security.

Framework obsługuje różne dowiązania danych

- JAXB 2.x,
- Aegis (2.1),
- Aegis Databinding (2.0.x),
- MTOM Attachments with JAXB
- SDO

By wygenerować dowiązanie SOAP 1.1 (na podstawie opisu wsdl bez sekcji z dowiązaniem, wystawionego pod wskazanym adresem wsdlurl):

```
wsdl2soap [[-?] | [-help] | [-h]] {-iport-type-name} [-bbinding-name] [-
doutput-directory] [-ooutput-file]
[-nsoap-body-namespace] [-style (document/rpc)] [-use (literal/encoded)] [-
v ] [[ -verbose ] |  [ -quiet ]] wsdlurl
```

By wygenerować dowiązanie SOAP 1.2 (na podstawie opisu wsdl bez sekcji z dowiązaniem, wystawionego pod wskazanym adresem wsdlurl):

```
wsdl2soap [[-?] | [-help] | [-h]] {-iport-type-name} [-bbinding-name] {-
soap12} [-doutput-directory] [-ooutput-file]
[-nsoap-body-namespace] [-style (document/rpc)] [-use (literal/encoded)] [
-v] [[ -verbose] | [-quiet]] wsdlurl
```

https://cxf.apache.org/docs/soap-12.html (opisano, jak zadeklarować dowiązanie obsługujące header)

By wygenerować klasy Java na podstawie opisu wsdl (https://cxf.apache.org/docs/wsdl-to-java.html):

```
wsdl2java HelloWorld.wsdl
```

```
Usage : wsdl2java -fe|-frontend <front-end-name> -db|-databinding <data-
binding-name>
-wv <wsdl-version> -p <[wsdl-namespace =]package-name>* -sn <service-name>
```

```
-b <binding-file-name>* -reserveClass <class-name>* -catalog <catalog-file-
name>
-d <output-directory> -compile -classdir <compile-classes-directory> -impl
-server
-client -clientjar <jar-file-name> -all -autoNameResolution -
allowElementReferences|-aer<=true>
-defaultValues<=class-name-for-DefaultValueProvider> -ant
-nexclude <schema-namespace [= java-package-name]>* -exsh <(true, false)> -
noTypes
-dns <(true, false> -dex <(true, false)> -validate -keep
-wsdlLocation <wsdlLocation> -xjc<xjc-arguments>* -
asyncMethods<[=method1,method2,...]>*
-bareMethods<[=method1,method2,...]>* -mimeMethods<[=method1,method2,...]>*
-noAddressBinding
-faultSerialVersionUID <fault-serialVersionUID> -exceptionSuper
<exceptionSuper>
-mark-generated -suppress-generated-date -maxExtensionStackDepth
<maxExtensionStackDepth>
-h|-?|-help -version|-v -verbose|-V -quiet|-q|-Q
-wsdlList <wsdlurl>
```

Projekt mavenowy

Aby w eclipse stworzyć serwis korzystające z Apache CXF wystarczy utworzyć projekt typu DynamicWeb i potem skorzystać z gotowych wizardów (podejście topDown lub bottomUp).

Jednak tak stworzony projekt będzie projektem eclipsowym. Dużo ciekawszym sposobem na rozpoczęcie pracy jest utworzenie odpowiednio sparamatryzowanego projektu mavenowego. Projekty mavenowe można wczytać do różnych środowisk programowania (nie tylko do eclipsa).

Projekt mavenowy można utworzyć na dwa sposoby: z linii komend lub poprzez wizard eclipse.

Utworzenie projektu mavenowego z linii komend

Należy wpisać w konsoli komendę:

```
$ mvn archetype:generate -Dfilter=org.apache.cxf.archetype
```

Na pojawiający się monit można odpowiedzieć jak niżej:

```
Choose archetype: 2
```

co oznacza *org.apache.cxf.archetype:cxf-jaxws-javafirst (Creates a project for developing a Web service starting from Java code)*

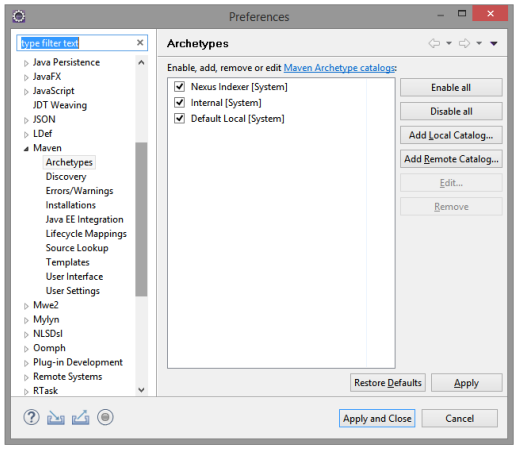
Kolejne monity i odpowiedzi:

```
Choose a version: – np. 99
Define value for property groupId: pl.edu.pwr.java
Define value for property artifactId: simpleService
Define value for property version: 1.0-SNAPSHOT
Define value for property package: pl.edu.pwr.java.simpleservice
```

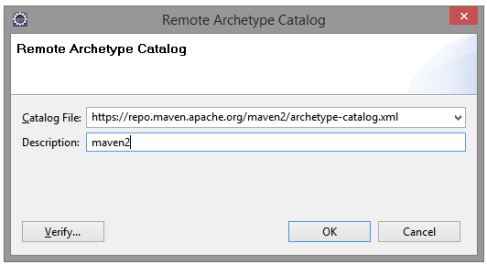
Po zatwierdzeniu wprowadzonych danych (Y) maven wygeneruje szkielet projektu.

Utworzenie projektu mavenowego w eclipse

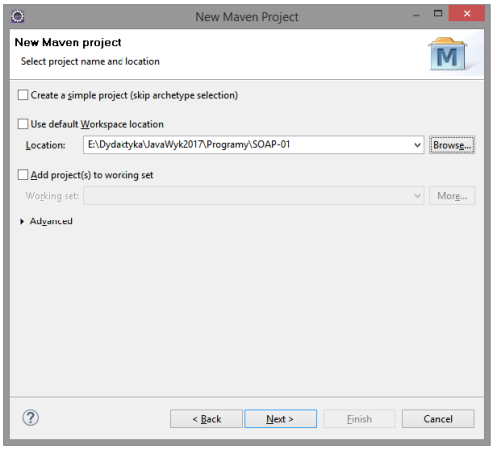
1. Otwórz okno preferencji:



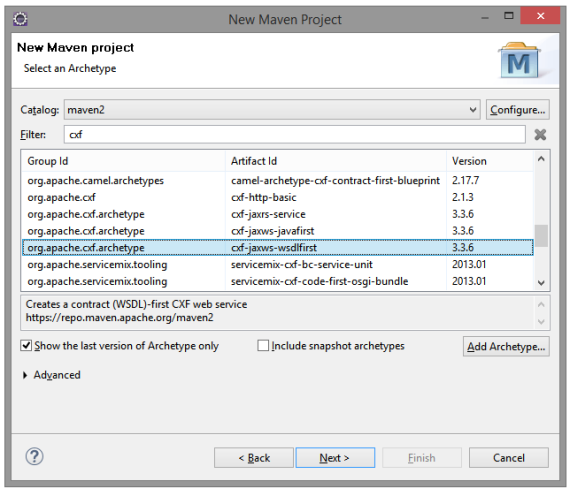
2. Dodaj zdalny katalog archetypów (po czym zatwierdź zmiany):



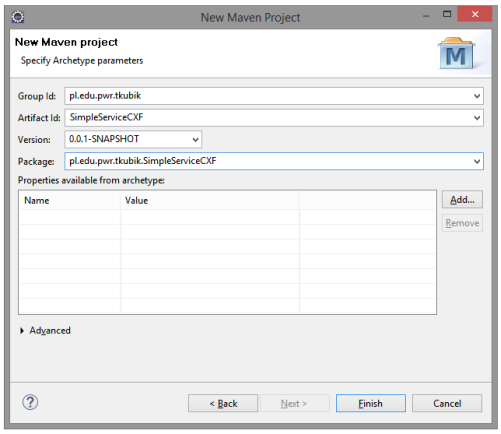
3. Utwórz nowy projekt mavenowy:



4. W kolejnym kroku wybierz odpowiedni archetyp (dla podejścia topDown jest to `cxf-jaxws-wsdlfirst`):



5. Potem zdefiniuj parametry projektu:



6. Po kliknięciu na Finish utworzony zostanie szkielet projektu.

Korekta projektu

Utworzony szkielet trzeba będzie zmodyfikować. Domyślnie (przy podejściu topDown) projekt będzie zawierał ustawienia i wygenerowane kody dla przykładu typu HelloWorld. Kody źródłowe można spokojnie usunąć, a pliki konfiguracyjne zmienić.

Pliki do zmiany to:

- `pom.xml`
- `\src\main\webapp\WEB-INF\web.xml`
- `\src\main\webapp\WEB-INF\bean.xml`

Ponadto należy dostarczyć plik z opisem WSDL mającego powstać serwisu:

- `src\main\webapp\WEB-INF\wsdl\simpleService.wsdl`

Całość można zrobić w następujacych krokach:

1. W `web.xml` należy zamienić kolejność dwóch linijek (z `sevlet-name` i `display-name`) oraz skonfigurować servlet (`url-pattern`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
        <context-param>
                <param-name>contextConfigLocation</param-name>
```

```
		<param-value>WEB-INF/beans.xml</param-value>
	</context-param>

	<listener>
		<listener-class>
			org.springframework.web.context.ContextLoaderListener
		</listener-class>
	</listener>

	<servlet>
		<display-name>CXF Servlet</display-name>
		<servlet-name>CXFServlet</servlet-name>
		<servlet-class>
			org.apache.cxf.transport.servlet.CXFServlet
		</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>CXFServlet</servlet-name>
		<url-pattern>/*</url-pattern>
	</servlet-mapping>
</web-app>
```

2. Należy zredagować plik wsdl z opisem serwisu (najlepiej w edytorze tekstu, graficzny edytor dostępny w eclipsie nie daje sobie ze wszystkim rady) i umieść go w katalogu `src/main/resources/wsdl/`

Zawartość tego pliku może być następująca (proszę uważać na przestrzenie nazw !!!!):

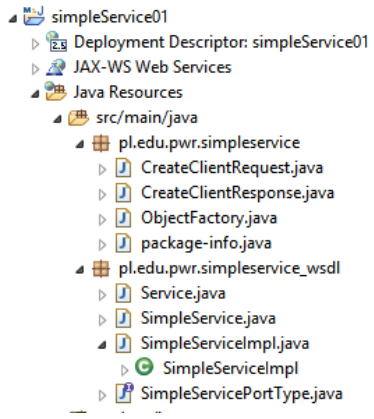
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
    targetNamespace="http://pwr.edu.pl/simpleService.wsdl"
    xmlns:tns="http://pwr.edu.pl/simpleService.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://pwr.edu.pl/simpleService.xsd">

  <wsdl:types>
    <xsd:schema targetNamespace="http://pwr.edu.pl/simpleService.xsd">
      <xsd:element name="CreateClientRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="surname" type="xsd:string"/>
            <xsd:element name="number" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="CreateClientResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="response" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="CreateClientRequestMsg">
    <wsdl:part element="xsd1:CreateClientRequest" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="CreateClientResponseMsg">
    <wsdl:part element="xsd1:CreateClientResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="SimpleServicePortType">
    <wsdl:operation name="CreateClient">
      <wsdl:input message="tns:CreateClientRequestMsg"/>
      <wsdl:output message="tns:CreateClientResponseMsg"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleServiceSoapBinding" type="tns:SimpleServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="CreateClient">
      <soap:operation soapAction="http://pwr.edu.pl/CreateClient"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="simpleService">
    <wsdl:port binding="tns:SimpleServiceSoapBinding" name="SimpleServicePort">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

3. W `pom.xml` należy poprawić parametry, by odzwierciedlały położenie źródeł generowanych z powyższego pliku wsdl przez mavenowy plugin:

```
<plugin>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-codegen-plugin</artifactId>
                <version>3.3.6</version>
                <executions>
                    <execution>
                        <configuration>
                            <sourceRoot>
                                src/main/java
                            </sourceRoot>
                            <wsdlOptions>
                                <wsdlOption>
                                    <wsdl>
                                       src/main/webapp/WEB-INF/wsdl/simpleService.wsdl
                                    </wsdl>
                                    <wsdlLocation>classpath:simpleService.wsdl</wsdlLocation>
                                </wsdlOption>
                            </wsdlOptions>
                        </configuration>
                        <goals>
                            <goal>wsdl2java</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

4. Po update projektu mavenowego utworzone zostaną katalogi z wygenerowanymi źródłami interfejsu serwisu oraz przykładem klienta (nazwy pakietów i klas będą wzięte z pliku wsdl).

Do tak wygenerowanych klas należy dołożyć klasę implementującą interfejs serwisu klienta i jeśli jest taka potrzeba – klasę publikującą serwis (klasa ta nie będzie potrzebna, jeśli serwis zostanie wdrożony na tomcacie).



Przykładowa implementacja serwisu:

```
package pl.edu.pwr.simpleservice_wsdl;

import pl.edu.pwr.simpleservice.CreateClientRequest;
import pl.edu.pwr.simpleservice.CreateClientResponse;

public class SimpleServiceImpl implements SimpleServicePortType {

          @Override
          public CreateClientResponse createClient(CreateClientRequest parameters) {
                    System.out.println(parameters.getName()+" "+
                                       parameters.getSurname()+" "+
                                       parameters.getNumber());
                    CreateClientResponse cr = new CreateClientResponse();
                    cr.setResponse("OK");
                    return cr;
          }

}
```

Podobny przykład opisano na stronie:
https://www.tutorialspoint.com/apache_cxf/apache_cxf_with_wsdl_first.htm

5. Aby wszystko zadziałało należy jeszcze wprowadzić zmiany do `beans.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

    <import resource="classpath:META-INF/cxf/cxf.xml" />

    <jaxws:endpoint id="CreateClient"
        implementor="pl.edu.pwr.simpleservice_wsdl.SimpleServiceImpl"
        serviceName="s:simpleService"
        xmlns:s="http://pwr.edu.pl/simpleService.wsdl"
        address="CreateClient"
        wsdlLocation="WEB-INF/wsdl/simpleService.wsdl">
        <jaxws:features>
            <bean class="org.apache.cxf.feature.LoggingFeature" />
        </jaxws:features>

    </jaxws:endpoint>
</beans>
```

Należy też zmodyfikować zawartość `pom.xml`. Poniże przedstawiono zestaw zależności w `pom.xml`, który pozwala na „redeployowanie” aplikacji na tomcacie.

```
<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"/>

<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</groupId>
    <artifactId>simpleService01</artifactId>
    <version>0.0.1-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>
    <dependencies>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-features-logging</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>5.1.14.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.14.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
    <scope>provided</scope>
        </dependency>

<!— jeśli serwis być opublikowany we własnej klasie, to musi istnieć kontener, np. jetty
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>3.3.6</version>
        </dependency> -->
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.cxf</groupId>
                <artifactId>cxf-codegen-plugin</artifactId>
                <version>3.3.6</version>
                <executions>
                    <execution>
                        <configuration>
                            <sourceRoot>
                                src/main/java
                            </sourceRoot>
                            <wsdlOptions>
                                <wsdlOption>
                                    <wsdl>
                                        src/main/webapp/WEB-INF/wsdl/simpleService.wsdl
                                    </wsdl>
                                    <wsdlLocation>classpath:simpleService.wsdl</wsdlLocation>
                                </wsdlOption>
                            </wsdlOptions>
                        </configuration>
                        <goals>
                            <goal>wsdl2java</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
        <pluginManagement>
            <plugins>
                <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>/simpleService</path>
                    </configuration>
                    <executions>
                        <execution>
                            <id>start-tomcat</id>
                            <goals>
                                <goal>run-war</goal>
                            </goals>
                            <phase>pre-integration-test</phase>
                            <configuration>
                                <path>/simpleService</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>1.8</source>
                        <target>1.8</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>
```

Działa to tak: najpierw trzeba skonfigurować i uruchomić samemu tomcata (wersja 8.5 lub wyżej, dla wersji 7,0 pojawiały się błędy). 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>
```

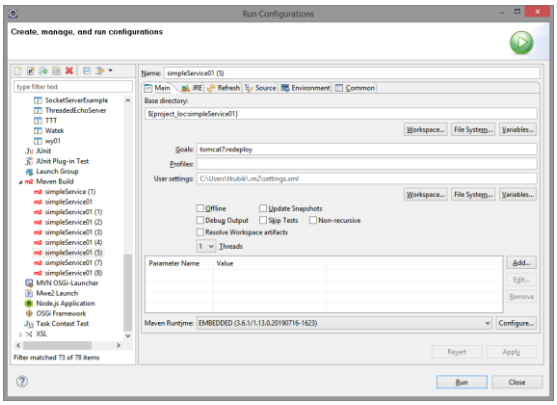
Poprzez ten węzeł plugin mavena dowie się, w jakiej sekcji pliku `~.m2\settings.xml` szukać ma parametrów uwierzytelniających do tomcata. W tam właśnie pliku powinien znaleźć się wpis:

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

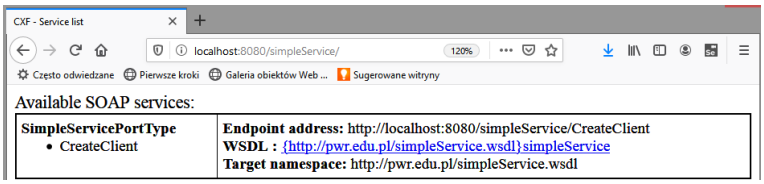
Aby skompilować i wdrożyć na tomcacie serwis wystarczy teraz wywołać:

`mvn tomcat7:redeploy`

(można również stworzyć konfigurację uruchomieniową w eclipse:



Po wdrożeniu aplikacji na tomcacie będzie można zobaczyć stronę z opisem endpointu (i linkiem do wsdl):



Serwis ten można odpytać uruchamiając narzędzie SoapUI (tworząc projekt soapowy, przekazując url do opisu wsdl działającego serwisu: `http://localhost:8080/simpleService/CreateClient?wsdl`)