PA165 - Lab session – WS-* Webservices
2.12.2014

Goals
a) set-up a WS-* webservice in Spring, b) understand "contract-first" development of web services.

Prerequisites
Netbeans 7.3.x, Tomcat 7, Java 7, Maven 3

Scenario
The project for this seminar uses Spring-WS to expose an endpoint to manage a series of Book(s) resources. You will be requested to add new functionality and change the configuration of the project.
The project uses the "contract-first" approach, so all the domain classes are generated from the xsd schema present in `src/main/resources/books.xsd` - all the domain classes will be recreated based on this file.
There is one integration test class in package `cz.muni.fi.pa165.soa.test`, you can use it to test the functionality you are implementing.

Task 1
In the IS you can find the initial application in the file **PA165-Fall2014-Seminar12spring-ws-seminar.zip**. Have a look at the application, explore the different parts. Check also that the application is runnable.
You should be able to import the project in Netbeans and run from it. Alternatively you can compile & run it from the command line from the root of the project:

```
module add maven-3.0.5
mvn clean install && mvn spring-boot:run
```

Upon successful execution you can find a published wsdl file at
http://localhost:8080/ws/books.wsdl

Look at this file and the parameters exposed in the `WebServiceConfig` class and the `books.xsd` schema.

You can use curl to interrogate the endpoint. Create one file called `request.xml` in your current directory, containing

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                          xmlns:ga="http://muni.cz/pa165/soa">
   <soapenv:Header/>
   <soapenv:Body>
      <ga:getBookRequest>
         <ga:title>The Hitchhiker's Guide to the Galaxy</ga:title>
      </ga:getBookRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Use curl from the same directory with `curl --header "content-type: text/xml" -d @request.xml http://localhost:8080/ws`

You should get similar response as the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
     <ns2:getBookResponse xmlns:ns2="http://muni.cz/pa165/soa">
      <ns2:book><ns2:title>The Hitchhiker's Guide to the Galaxy</ns2:title>
      <ns2:isbn>0345391802</ns2:isbn>
      <ns2:price>55</ns2:price>
      <ns2:author>
            <ns2:name>Douglas</ns2:name><ns2:surname>Adams</ns2:surname>
      </ns2:author>
      <ns2:availability>ORDERED</ns2:availability>
      </ns2:book>
    </ns2:getBookResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From now on, you can use the tests within the application to test requests & responses and their payloads. For this reason, look also that tests in `cz.muni.fi.pa165.soa.test` work as expected, there is one test that tests exactly the same request and response pair as the one given with the curl command.

Task 2

It would be better to add logging to the application, so that we can log the different SOAP messages.
Add a file called `logback.xml` in your `src/main/resources` folder with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <logger name="org.springframework.web" level="DEBUG"/>
    <logger name="org.springframework.boot" level="DEBUG" />
    <logger name="org.springframework.ws.server.MessageTracing.sent"
level="TRACE" />
    <logger name="org.springframework.ws.server.MessageTracing.received"
level="DEBUG" />
 </configuration>
```

Try to play a bit with `MessageTracing` different logging levels and see the difference when passing from `TRACE`, `DEBUG` and `INFO` when the endpoint handler methods are invoked (you can do this either with curl or with the tests).

Task 3

It is a good idea to validate the schema of the responses we are sending out. For this, you can add an interceptor. Spring-WS gives you the possibility of adding interceptors for different purposes (e.g. security). We will use in this case the `PayloadValidatingInterceptor`.

Open your WebServiceConfig class and create a new interceptor adding the following method:

```
    @Bean
    public PayloadValidatingInterceptor myPayLoadInterceptor() {
      PayloadValidatingInterceptor interceptor = new
                                  PayloadValidatingInterceptor();
      interceptor.setXsdSchema(this.booksSchema());
      interceptor.setValidateRequest(true);
      interceptor.setValidateResponse(true);
      return interceptor;
  }
```

Then we add the interceptor to the list of endpoint interceptors.

```
 @Override
    public void addInterceptors(List<EndpointInterceptor> interceptors) {
        interceptors.add(this.myPayLoadInterceptor());
    }
```

To give an example, without validation the following request will pass through to the endpoint (even if <name> was not defined in the schema):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ga="http://muni.cz/pa165/soa">
<soapenv:Header/>
      <soapenv:Body>
            <ga:getBookRequest>
                  <ga:title>The Hitchhiker's Guide to the Galaxy</ga:title>
                  <ga:name>The Hitchhiker's Guide to the Galaxy</ga:name>
            </ga:getBookRequest>
      </soapenv:Body>
</soapenv:Envelope>
```

After the addition of the interceptor, the schema will be validated, so in that case you will get a response with a SOAP Fault, like the following:

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
      <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Client</faultcode>
            <faultstring xml:lang="en">Validation error</faultstring>
            <detail><spring-ws:ValidationError xmlns:spring-
            ws="http://springframework.org/spring-ws">cvc-complex-type.2.4.d:
            Invalid content was found starting with element 'ga:name'. No child
            element is expected at this point.</spring-ws:ValidationError>
            </detail>
      </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You can read more about different interceptors in Spring-WS documentation: http://docs.spring.io/spring-ws/docs/2.2.0.RELEASE/reference/htmlsingle/#server-endpoint-interceptor

Task 4

In general, there is a problem with the current implementation: the method `getBookByAuthorNameAndSurname(...)` returns one book, but authors can have many books.
Change the behaviour by returning a list of books. This implies that the schema needs to be changed so that more books are returned by the endpoint (*hint: you can use minOccurs="0" maxOccurs="unbounded" in the element definition of book in the response – remember also the read comments in the new generated class to use it!*).

Have passing tests for your new implementation.

Task 5

When a book that does not exist in the service is requested, we want to return a SOAP Fault to the requestor. Currently, if we ask for a book that does not exist in the catalogue, we will get the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
      <SOAP-ENV:Body>
            <ns2:getBookResponse xmlns:ns2="http://muni.cz/pa165/soa"/>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The first thing you could try is to throw a `RuntimeException()` in your endpoint handler method. If you change your public `GetBookResponse getBook(...)` method in `BookEndpoint` class as follows

```
[...]
Book book = bookRepository.getBookByTitle(request.getTitle());

            if (book==null){
                throw new RuntimeException("Book " + request.getTitle() + " not
found.") ;
            }

[...]
```

The SOAP response will now look as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Server</faultcode>
                  <faultstring xml:lang="en">Book Alice in Wonderland not
                  found.</faultstring>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This is similar to the response we would like to provide in such cases. We need a more proogrammatic way to handle exceptions. In our case, we will use the `SoapFaultAnnotationExceptionResolver` and using the `@SoapFault` annotation.

So we need to create an Exception and then map to the soapfault:

```
@SoapFault(faultCode = FaultCode.SERVER, faultStringOrReason = "Book not found."
)
public class BookNotFoundException extends RuntimeException {
        public BookNotFoundException(String bookTitle) {
            super("Book not found '" + bookTitle );
      }
}
```

now we will throw the exception with

```
      if (book==null){
                  throw new BookNotFoundException(request.getTitle());
           }
```

The response will be like the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Server</faultcode>
                  <faultstring xml:lang="en">Book not found.</faultstring>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You should have created one test for the case in which there are no books to be returned. For this test, use `serverOrReceiverFault()` to test for the specific fault reason.
Usually you will need to map also `SoapFaultAnnotationExceptionResolver` as a bean, but in Spring-boot standard configuration for Spring-WS. You can read about exceptions in Spring-WS at http://docs.spring.io/spring-ws/docs/2.2.0.RELEASE/reference/htmlsingle/#server-endpoint-exception-resolver

Task 6

Similarly to `books.xsd`, create another schema `customers.xsd` in which you can report different properties such as name, surname, address for one customer. Each customer will have a list of books (defined in `books.xsd`).
Expose another endpoint that returns the list of customers with the books that have been leased.

Task (Extra)

If you have completed all the tasks, you can look at how to consume a web service, you can adapt a client application from the tutorial:
http://spring.io/guides/gs/consuming-web-service/