### **Creating a SOAP Service Using Spring Boot**

V3 2025 (added IntelliJ)

### **Licence**



This work by Duncan Mullier at Leeds Beckett University is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International Licence](http://creativecommons.org/licenses/by-sa/4.0/deed.en_GB).

### **Contents**

[Creating a SOAP Service Using Spring Boot](#_s20m5nws8o6s)

[**Licence**](#_d6rj2kee8392)

[Contents](#_s20m5nws8o6s)

[**Introduction**](#_ljnqxqe8p3ex)

[**Creating A Spring Boot SOAP Service**](#_bf2iy84puv26)

[**1 Create a Spring Boot Project**](#_epi6wb6oi0gp)

[**2 Add Extra dependencies to Maven for WSDL support.**](#_1c8bmb3ua8dq)

[**3 Create a service description file (XSD).**](#_dfr1f6a0zmgf)

[**5 Create the Domain Classes**](#_myg5ymxuch5k)

[**6 Create Our Country Repository**](#_5csxkg9d5pmx)

[**7 Create a Country Service Endpoint**](#_puieswqlddsf)

[**9 Running the Service**](#_thqtzlrkic8f)

[**10 Examining your service**](#_azqhremqir41)

### Introduction

The labs are written to be informative and, in order to aid clarity, instructions that you should actually execute are generally written in this colour. Instructions asking you to record something in your logbook are generally written in this colour.

In this tutorial we are going to create a SOAP Service, which we can use in another tutorial, using Sprint Boot. A service lives on the cloud and provides functionality that we can use in our programs. This is an order of magnitude more flexible that importing a library into our code. Services are also in control of the provider, without this they may not offer it at all, think Amazon, or Ebay.

I have also created a [tutorial which shows you how to create a service WITHOUT using Spring Boot](https://docs.google.com/document/d/1iJh8Uoj8aNDLA7O_L-8W58KJHuG79Ao0ENO7OOwLoqU/edit?usp=sharing). I urge you to look at that too to see what you are gaining here by using Spring Boot (because it is a royal pain without it).

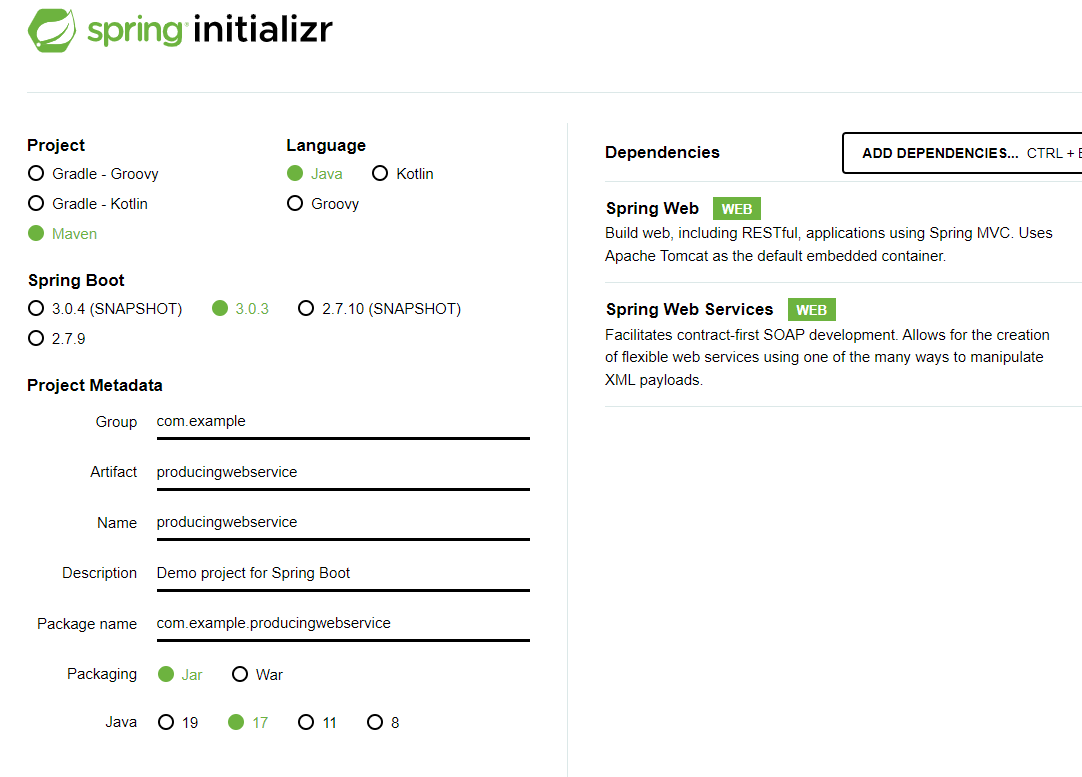
You can create a Spring Boot SOAP project using the command line, Eclipse, or IntelliJ. For Eclipse ensure you have it set up for Spring Boot see [The previous Spring Boot lab sheet.](https://docs.google.com/document/d/1P_gr5rboWSnQCkCXBNEU7PkgsJls-zLiAJEkKMbfUfE/edit#heading=h.dzfvi8cdmzi) For IntelliJ you will need the Ultimate Edition, [which you can get with a student licence](https://www.jetbrains.com/community/education/#students).

## Creating A Spring Boot SOAP Service

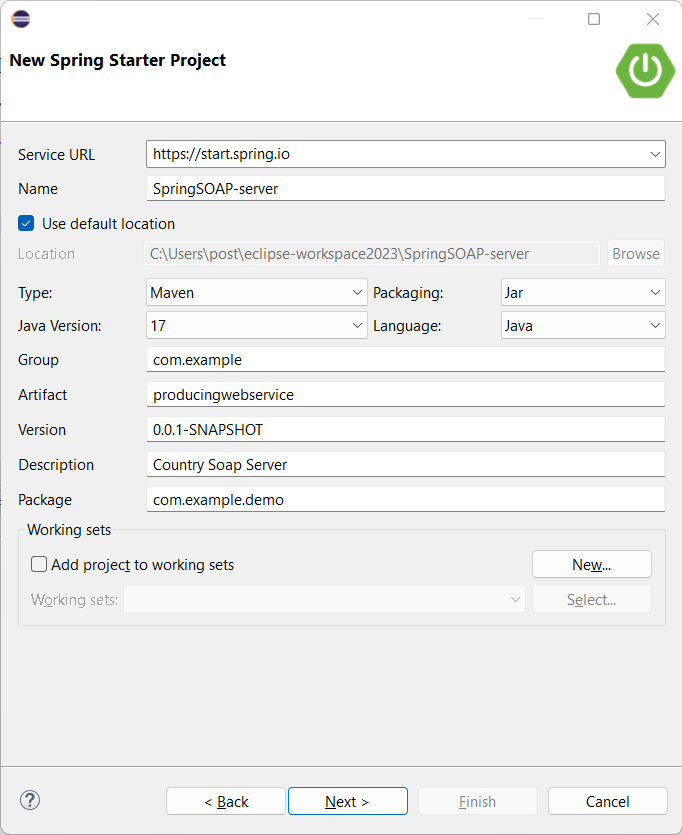
### 1 Create a Spring Boot Project

In your IDE or visit [The Spring Initilizr](https://start.spring.io) for a manual installation.

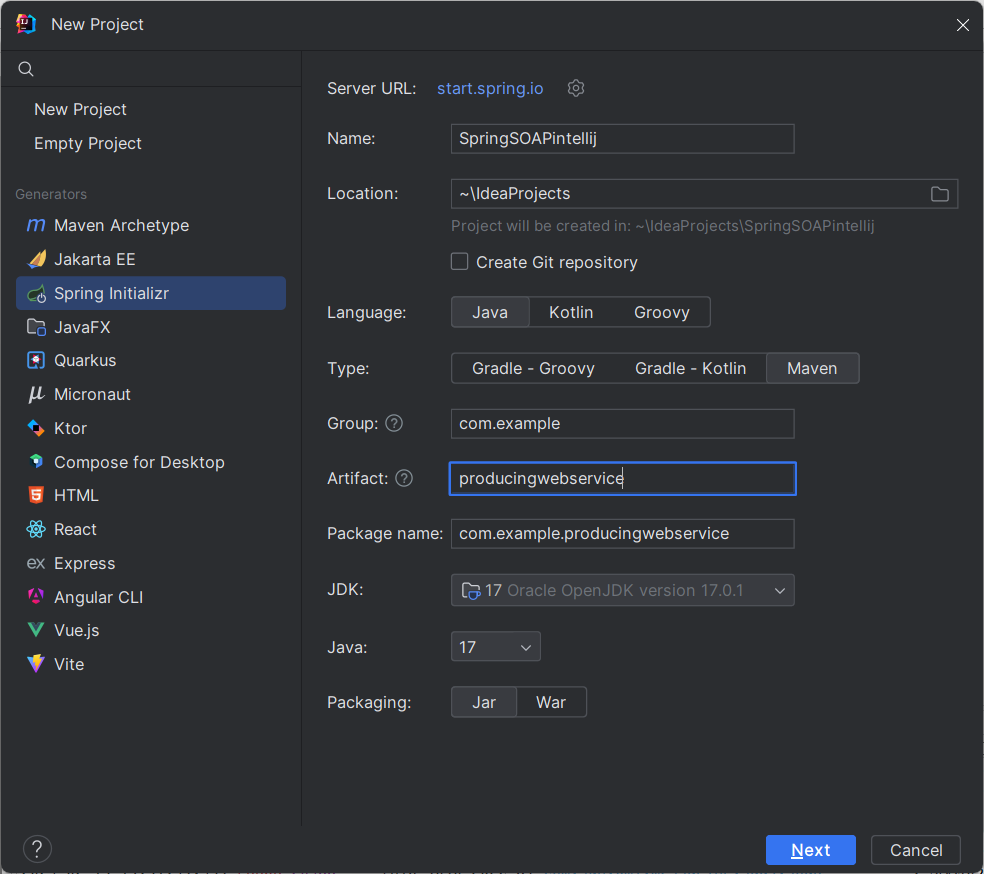
Select the options shown below.

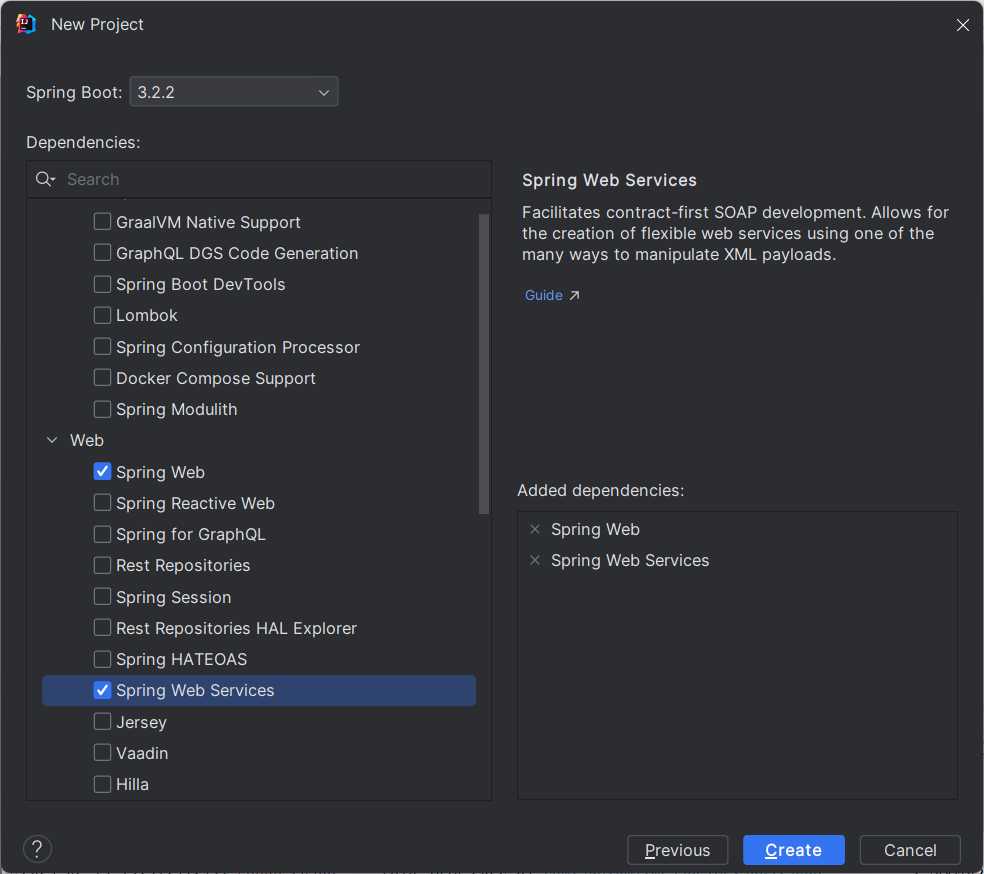


Or in Eclipse



Or In IntelliJ create a new->Spring Initializr Project. Of new->Spring Boot





Whichever way you choose, make a note in your logbook and show a screenshot of the project options.

The Group ID is the package it will put the project in. i.e. above it has a very generic com.example, so it will be in <yoursource>\com\example

If I were doing it properly I’d put it in a packaged named after my project and id address, so:

uk.ac.leedsbeckett.mullier.sesc might be better. It doesn’t really matter now, but that’s what it’s for.

The artefact ID is the name of your application.

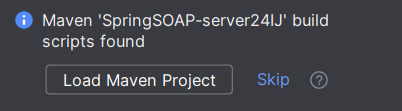
Here we will use the Maven build system. Maven and Gradel are build systems that handle all the dependencies. We could just as easily have used Gradel but I had to choose one and it was Maven. [If you are interested in more detail on Maven then see this extra worksheet.](https://docs.google.com/document/d/19_LRZNVyCZ9nBIGrBooabmBJaRKCe7MBzpN5RpN-tKA/edit?usp=sharing)

The application can be packaged as a Jar or War file. We will use Jar. War is generally used for web applications.

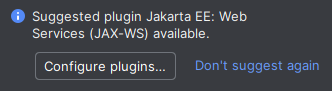
Notice that I have added two dependencies, ***Spring Web and Spring Web Services***. In Eclipse and IntelliJ this is done on the next page of the dialogue. Also, the Version is shown here in Eclipse as 0.0.1-SNAPSHOT (Snapshot means it is not a final release). The version is also selected on the next page in Eclipse but not at all on the web page or IntelliJ. This version is referring to the version of our software that we are building and obviously since we have just created the project it is the lowest version possible, this is why the web and IntelliJ don’t give us an option, they just set it to 0.0.1-Snapshot, which we can see when we open the resulting pom.xml file in our project.

Select the highest version that isn’t a Snapshot for the version of Spring Boot. (as of now 3.2.2).

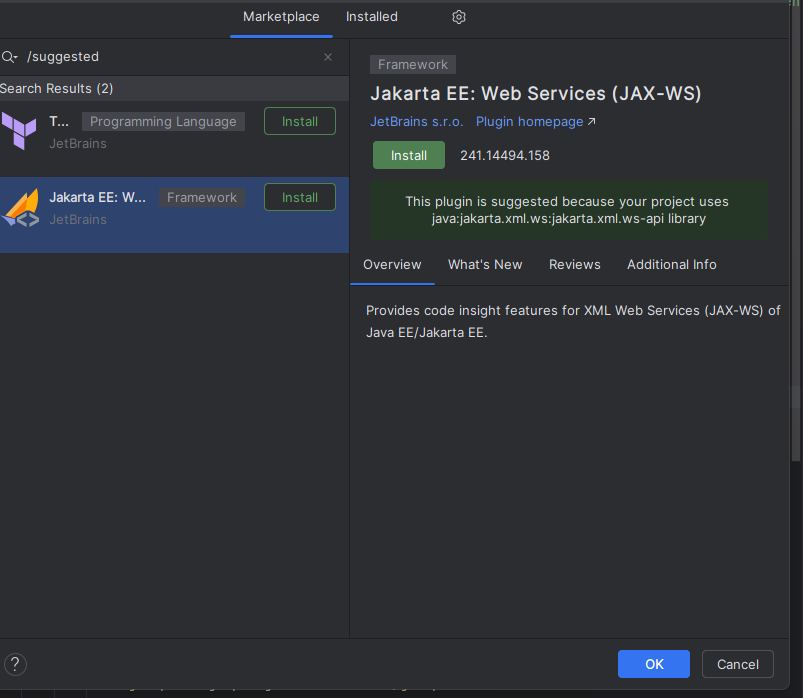
If using IntelliJ it will detect it is a Maven project and a dialogue will pop up. If for some reason it doesn’t pop up, right click on pom.xml and select Maven->Reload Project



Click to load the Maven Project.

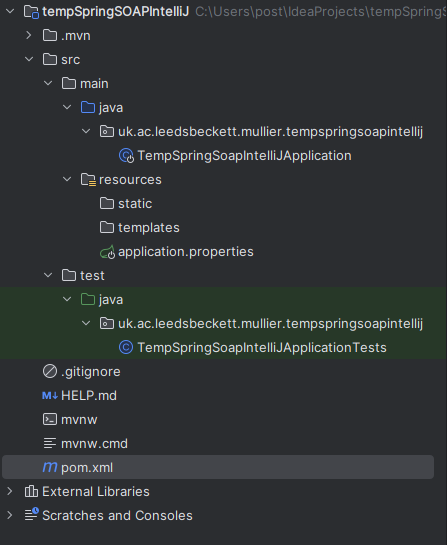


If this pops up, accept it.



Select Jakarta EE and then Install.

Once the project has been created you will have the following structure (shown in IntelliJ).



It has created a structure based on the information we gave it. Note here I gave it a different GroupID and it has created a package based on it called uk.ac.leedsbeckett.mullier (this is a better GroupID and I want to make that point now, I have used com.example in the rest of the example to keep it generic but generally you should use a GroupID that reflects your development).

1 It has created src/main and then put a package based on our GroupID and put a generic application java file in there.

2 It has created a mirror test directory for our unit testing (which is also SESC topic).

3 It has created an application.properties file. This is where we can change the port setting etc of the project (This is discussed in section 10).

4 It has created a src/main/resources directory. This is where non-java files go that are to do with our project.

5 It has created a pom.xml file. This is the Maven file (it wouldn’t be there if we’d used Gradel) that tells the project what dependencies and plugins it needs. We are going to be very interested in this file.

### 2 Add Extra dependencies to Maven for WSDL support.

WSDL (Web Service Description Language) is how the service we are creating describes its services to the outside world. It is an XML file that tells clients how to use it.

See the [Maven tutorial](https://docs.google.com/document/d/19_LRZNVyCZ9nBIGrBooabmBJaRKCe7MBzpN5RpN-tKA/edit?usp=sharing), but we are going to add this to the pom.xml file of the project.

|  |
| --- |
| <dependency>  <groupId>wsdl4j</groupId>  <artifactId>wsdl4j</artifactId>  </dependency> |

You’ll see in the pom.xml file there are several dependencies between

<dependencies>

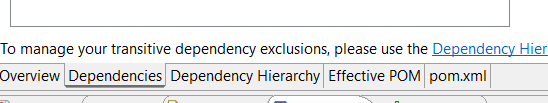
</dependencies>

In Eclipse you can add manually or right click on the pom file in the project explorer and select Maven->Add Dependency. In IntelliJ right click on the background of the pom.xml file and select Generate->Dependency.

In IntelliJ you can press Alt-Enter on the open pom.xml and do it from the context menu.

Type Wsdl4j for the groupId and artefactId.

If you do this manually by adding it to pom.xml make sure you put the wsdl4j dependency between these tags. You can also do this in Eclipse by pasting it in, or use the Eclipse interface. Use the interface by clicking on pom.xml in the package explorer and then clicking on the “Dependencies” tab.



This tells Maven to bring in everything necessary to produce and parse WSDL files into our project. Note that this gives two advantages, one, we don’t have to write a WSDL parser (this is like importing a library in Java from a third party source). Two, we don’t have to know where the library is or what files it needs.

If you manually add to the pom.xml in IntelliJ it will show up in red, as it hasn’t loaded it. You should see the  icon. Click this to load the new dependency.

If you still have a problem, it will throw an error one of which is a clickable item that says “try to run with -u”, click this and it should update all of the repos.

Make a note of this step in your logbook.

### 3 Create a service description file (XSD).

A file with the XSD [file extension](https://www.lifewire.com/what-is-a-file-extension-2625879) is most likely an XML Schema file; a text-based file format that defines validation rules for an [XML](https://www.lifewire.com/what-is-an-xml-file-2622560) file and explains the XML form.

Spring will take this file, which describes our service, and create a WSDL description of our service that we can give to client applications.

This file goes in the src/main/resources directory.

|  |
| --- |
| <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://spring.io/guides/gs-producing-web-service"  targetNamespace="http://spring.io/guides/gs-producing-web-service" 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> |

The bits of interest are the “getCountryRequest” and “getCountryResponse”. Clients will send an xml schema resembling the former, with a country to query the service and receive the latter back, with the result. This file is telling prospective clients how to set a schema up to query our service and what schema they will receive back. When you create a client this schema will configure the skeleton code to send and receive these requests.

Create a new blank file called “countries.xsd”. When you call it an xsd file it may show a scheme view in Eclipse. Below it you will see “design” and “source”. Click source to paste the above in and save it.

Remember that this is a very simple example and you can imagine that if we were to do this for real then we’d have a lot more data and probably use a database for it.

Make a note of this step in your logbook.

### 5 Create the Domain Classes

We need the code to actually perform the task (here return the data about a country). Since we have formally specified how this data is to be received and returned, it makes sense to generate the blank classes from this (to avoid mistakes, i.e. we could write the classes ourselves based on the description). We can use Maven to do this but it requires a plugin. So it is back to the pom.xml file.

You will see a <plugins> section. Put the following inside it.

|  |
| --- |
| <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>  </configuration>  </plugin> |
|  |

Once you have added this you may see a syntax error in your project.

In eclipse right click on your project and select maven->update project

or

In IntelliJ this will appear  click it and the project will be rebuilt, if not right click on pom.xml and select Maven->reload project

or

On the command line you can issue mvn:install

Notice that this plugin references the countries.xsd file that you created above.

If you are using later versions of Java you will still get syntax errors because the library that parses the annotations has been removed. You can add them as a Maven dependency.

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>javax.annotation-api</artifactId>

<version>1.3.2</version>

</dependency>

See <https://stackoverflow.com/questions/52502189/java-11-package-javax-xml-bind-does-not-exist>

The syntax errors should disappear.

If you are still having problems at this stage it can be because of your installed version of the JRE. Some require an additional dependency. Add in:

<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.4.0-b180830.0359</version>

</dependency>

Ref: <https://stackoverflow.com/questions/61177573/javax-xml-bind-cannot-be-resolved>

Make a note of this step in your logbook.

### 6 Create Our Country Repository

Create a package called com.example.producingwebservice and create a file called CountryRepository.java inside it.

(In Eclipse right click project->New->Package (IntelliJ will have already created one).

and then right click com.example.producingwebservices->New->class).

Note if you’ve used a more sensible GroupID you will have a different package, just don’t copy the package statement below and use the appropriate one for your package.

Paste the following code into it.

|  |
| --- |
| package com.example.producingwebservice;  import javax.annotation.PostConstruct;  import java.util.HashMap;  import java.util.Map;  import io.spring.guides.gs\_producing\_web\_service.Country;  import io.spring.guides.gs\_producing\_web\_service.Currency;  import org.springframework.stereotype.Component;  import org.springframework.util.Assert;  @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);  }  } |

Here the @ annotations are telling the Spring Framework what you want it to do. @componet is telling Spring that this class is a Bean, a Bean being a class that you want Spring to manage (instantiate, inject etc). @PostConstruct is telling Spring that you want the following code called once, when the Bean is instantiated, i.e. like a constructor. If you used an actual constructor then Spring would not be able to manipulate it so easily.

In Eclipse right click on the project and select Maven-Update Project

In IntelliJ will need you to manually update the Maven Project

Select “Generate Sources and Update Folders” from the Maven Menu.This should get rd of syntax errors.

If you get syntax errors on the @postconstruct decoration (likely on a lab machine), then add the following dependency.

<dependency> <groupId>javax.annotation</groupId> <artifactId>javax.annotation-api</artifactId> <version>1.3.2</version> </dependency>

Ref: <https://stackoverflow.com/questions/53690136/the-import-javax-annotation-postconstruct-cannot-be-resolved>

If this doesn’t fix it add these dependencies as well.

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-core</artifactId>

<version>2.3.0</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.3.0</version>

</dependency>

<https://stackoverflow.com/questions/52502189/java-11-package-javax-xml-bind-does-not-exist>

Make a note of this step in your logbook.

### 7 Create a Country Service Endpoint

An endpoint is the way a client interacts with a service, usually via HTTP. This is a plain old java object (POJO) which is what will be called by the client. It has several Spring Boot annotations, which tell Spring Boot what to do.

|  |
| --- |
| package com.example.producingwebservice;  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;  import io.spring.guides.gs\_producing\_web\_service.GetCountryRequest;  import io.spring.guides.gs\_producing\_web\_service.GetCountryResponse;  @Endpoint  public class CountryEndpoint {  private static final String NAMESPACE\_URI = "http://spring.io/guides/gs-producing-web-service";  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;  }  } |

In your producingwebservice package create a class/java file called CountryEndpoint and copy the above code into it.

The [@Endpoint](https://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/ws/server/endpoint/annotation/Endpoint.html) annotation registers the class with Spring as a potential candidate for processing incoming SOAP messages.

The @Autowired annotation tells Spring to call the setter with the instance of CountryRepository when the CountryEndpoint is created/injected.

The [@PayloadRoot](https://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/ws/server/endpoint/annotation/PayloadRoot.html) annotation is then used by Spring to pick the handler method, based on the message’s namespace and localPort.

The [@RequestPayload](https://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/ws/server/endpoint/annotation/RequestPayload.html) annotation indicates that the incoming message will be mapped to the method’s request parameter.

The [@ResponsePayload](https://docs.spring.io/spring-ws/sites/2.0/apidocs/org/springframework/ws/server/endpoint/annotation/ResponsePayload.html) annotation makes Spring map the returned value to the response payload.

Make a note of this step in your logbook.

8 Create A Web Service Bean

Create WebServiceConfig.java in the producingwebservice package and copy the following into it.

|  |
| --- |
| package com.example.producingwebservice;  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://spring.io/guides/gs-producing-web-service");  wsdl11Definition.setSchema(countriesSchema);  return wsdl11Definition;  }  @Bean  public XsdSchema countriesSchema() {  return new SimpleXsdSchema(new ClassPathResource("countries.xsd"));  }  } |

The elements highlighted in red are used to generate the WSDL description of the service. It will take the form:

[http://<host>:<port>/ws/countries.wsdl](about:blank)

Make a note of this step in your logbook.

### 9 Running the Service

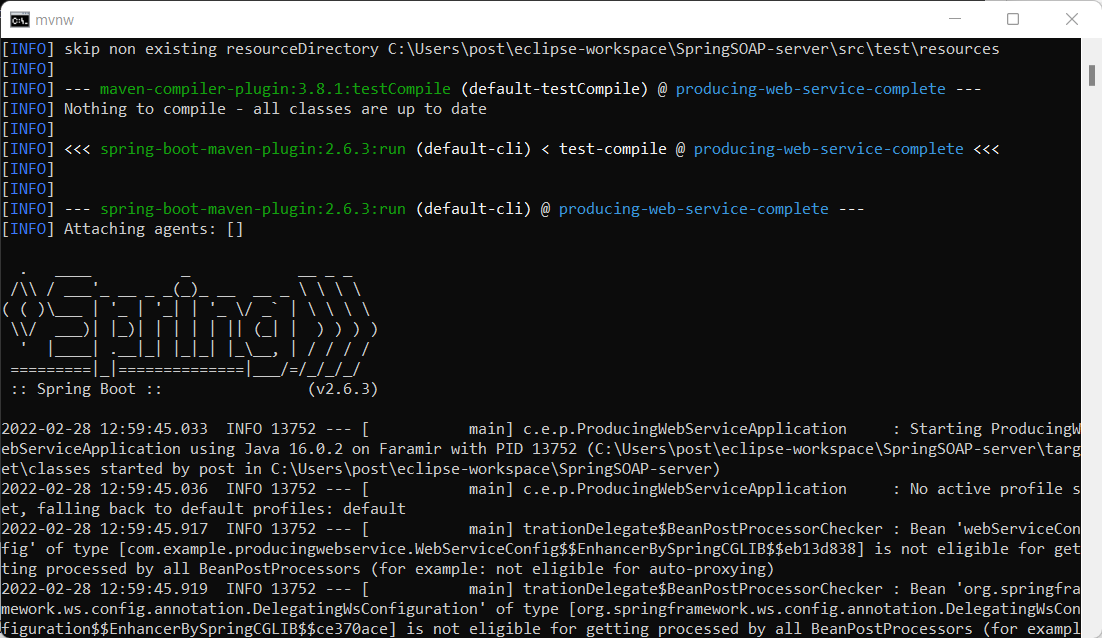
To run using Eclipse, right click on your project and select RunAs-> “Spring Boot App”.

To run using IntelliJ click the run green triangle icon on the top menu bar.

To run using Maven from the command line, open a command prompt, go to the directory or your project, the one with the pom file and type mvnw spring-boot:run

You can also run the jar file that Maven creates with mvnw clean package

You can then run the java file it creates in the target directory with java -jar target/<your project>.jar



To see that it is working goto <http://localhost:8080/ws/countries.wsdl>

You should get an output of the WSDL schema for your service.

You can stop your service in Eclipse by clicking the red “stop” square on the bottom right. If you are running under the command line you will have to close the command window.

NOTE on the some versions of Eclipse with the current version of Spring Boot there is an inconsistency which sometimes results in Eclipse not finding the wsd file. This can cause an exception when Spring Boot runs and/or a “White Label” error message when you try and connect to the server. The project is ok and you can get it to work from the command line. Maven has put everything there you need (including itself).

You can build the application with (although Eclipse should have already done it and there should be a jar in the target directory):

mvnw clean package

You can then run it with.

mvnw spring-boot:run

Or run the jar file directly with:

java -jar target/producingwebservice-0.0.1-SNAPSHOT.jar

(or whatever your jar file is called in your target directory).

Note that the 0.0.1-SNAPSHOT comes from a tag in our pom.xml file.

<version>0.0.1-SNAPSHOT</version>

It is the version of OUR server. Here a very early unreleased (snapshot) version.

[For an explanation of versions and snapshots see this link.](https://www.baeldung.com/maven-snapshot-release-repository#:~:text=A%20snapshot%20version%20is%20a,artifact%20version%20they're%20consuming.)

It is worth taking a breath here and thinking on what you have accomplished. The jar file in the target directory is a full web service application, including Tomcat server. A single file can be given to someone and they can just run it!

Make a note of this step in your logbook.

### 10 Examining your service

Run your service

You can examine your service using [SoapUI](https://www.soapui.org/downloads/soapui/). Eclipse also has a service explorer, but it isn’t that friendly and you can also get a response from a web browser.

The WSDL file, which is the descriptor of your service, is formed from the XSD XML file that you created in step 4 and the domain classes from step 5. In our WebServiceConfig.java we mvnw clean package called our servlet “ws” and it will be running on a Tomcat server under the default port of 8080 (unless we’ve changed the port, see the text output in the console to see which port it is running under), then we can get at our WSDL file:

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

If you did want to change the default port then one way of doing it is to create

/src/main/resources/application.properties

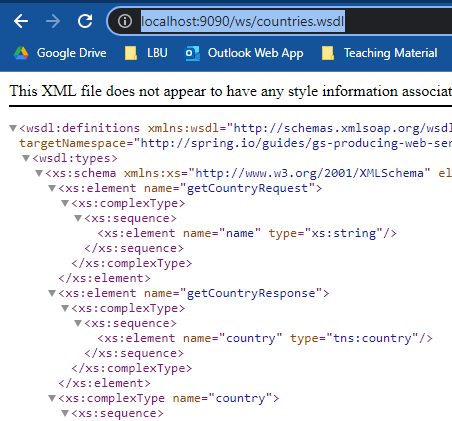
And inside:

server.port=8888

If you put zero for the port it will allocate a port and you can see what it is when your application starts.

For further information see <https://mkyong.com/spring-boot/spring-boot-how-to-change-tomcat-port/>

Here is a screenshot of my service running on port 9090:

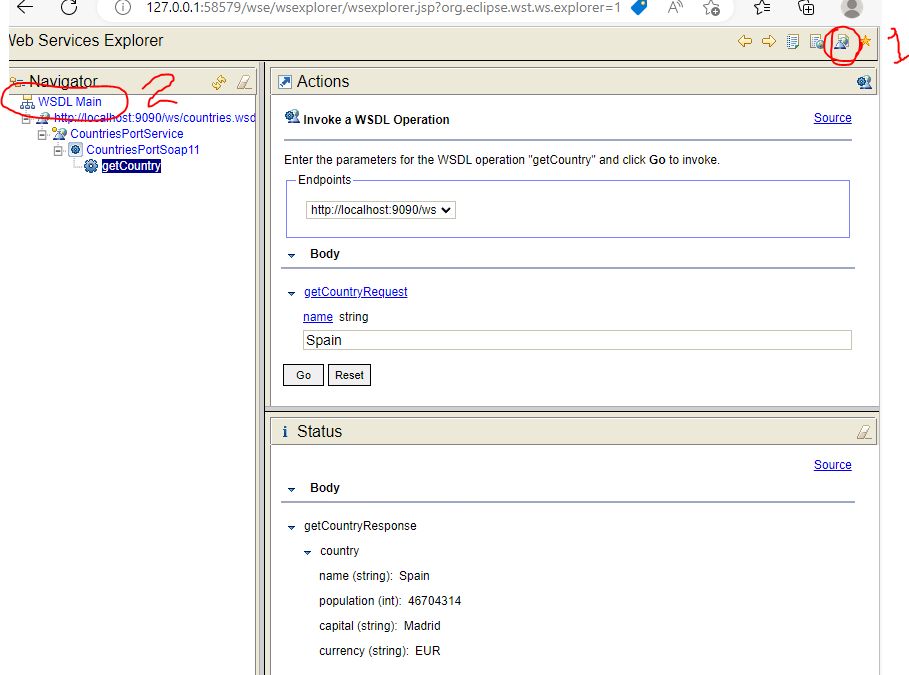


To use the Eclipse Service Explorer goto run->Launch the Web Service Explorer

Click the icon on the top right, next to the star, Launch the WSDL page.

On the left click “WSDL Page” and a text field will appear.

Paste the url of your server into here.



Here you can see that I’ve asked the Service about Spain and it has responded with its information about Spain. If you click “source” you will see the full xml envelope.

Soap UI can be downloaded and installed on your machine or a lab machine.

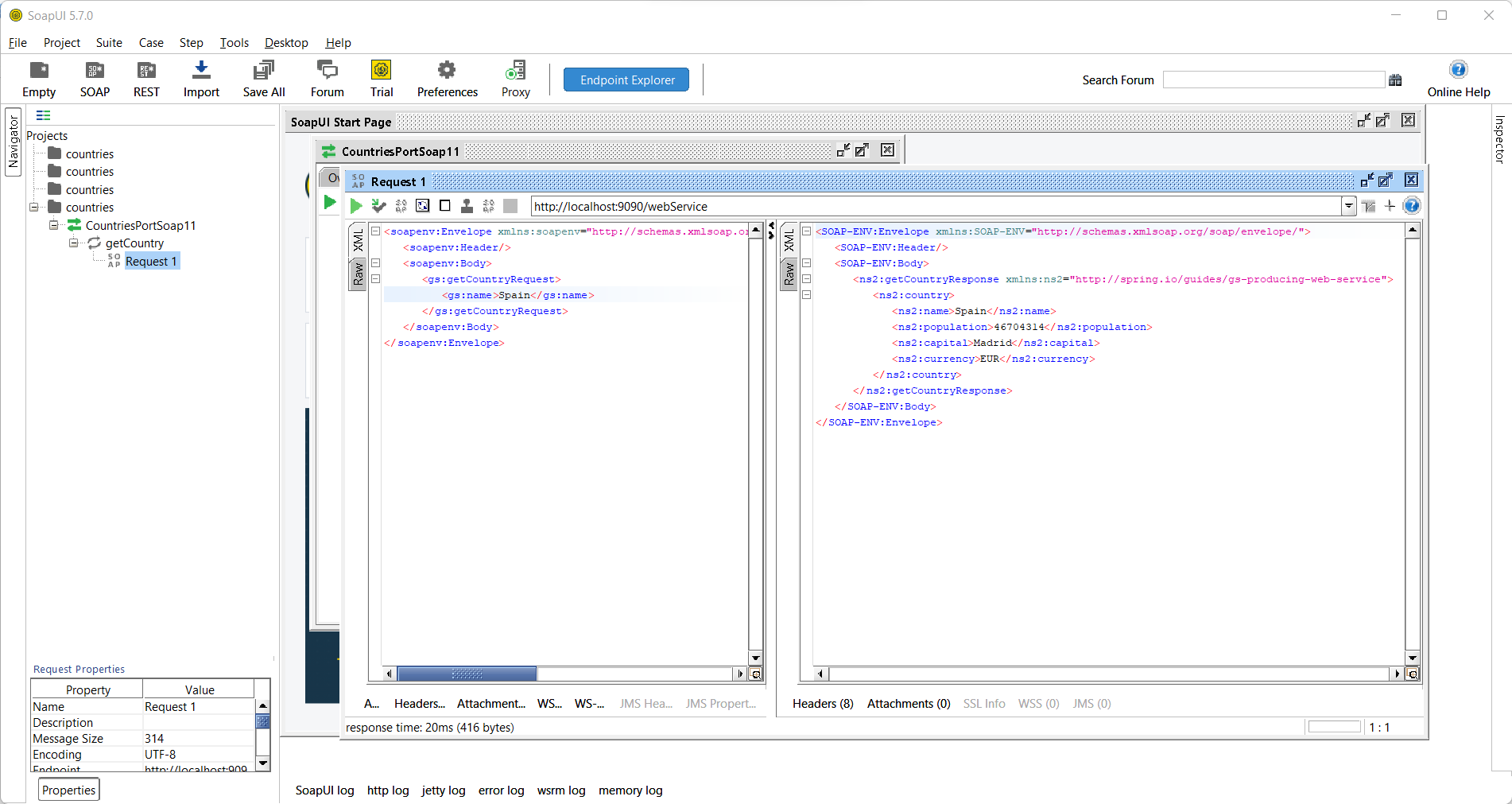
[Download REST & SOAP Automated API Testing Tool | Open Source | SoapUI](https://www.soapui.org/downloads/soapui/)

Download the Open Source one.

In SoapUI create a new SOAP project and enter <http://localhost:8080/ws/countries.wsdl>

You have given it your WSDL file (note this has come from the service itself but you could save this as a text file and give it that).

Your service will be loaded and you will be able to interact with it.



Expand your service “countries” and you will see your “getCountry” endpoint. Expand it until you see “Request1”, an XML schema will open up (make sure you click XML and not RAW on the side”.)

|  |
| --- |
| <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:gs="http://spring.io/guides/gs-producing-web-service">  <soapenv:Header/>  <soapenv:Body>  <gs:getCountryRequest>  <gs:name>?</gs:name>  </gs:getCountryRequest>  </soapenv:Body>  </soapenv:Envelope> |

Change the “?” to “Spain” and click the green “play” arrow. You will be a response! (Make sure you click XML again).

|  |
| --- |
| <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  <SOAP-ENV:Header/>  <SOAP-ENV:Body>  <ns2:getCountryResponse xmlns:ns2="http://spring.io/guides/gs-producing-web-service">  <ns2:country>  <ns2:name>Spain</ns2:name>  <ns2:population>46704314</ns2:population>  <ns2:capital>Madrid</ns2:capital>  <ns2:currency>EUR</ns2:currency>  </ns2:country>  </ns2:getCountryResponse>  </SOAP-ENV:Body>  </SOAP-ENV:Envelope> |

You can see that we asked it about Spain and it has told us about Spain’s capital city and population. OK our service isn’t very interesting or comprehensive but it is a service!

Make a note of this step in your logbook.

Expand the number of countries that the service knows about. You should have enough information by now to know how to do this, get it to run again and interrogate it.

Make a note of this step in your logbook showing how you have expanded the service.

The next session will look at creating a SOAP client, whereby we can use a service in our programs. You can imagine that there are many services that can be used, for example Amazon services.

## If All Else Fails…

I have a project on GitHub that you can clone.

<https://github.com/LBU-SESC/SpringBootSOAPserver.git>

In Eclipse go to the Git Perspective

Select “Clone a Repository and add to this view” or “Clone a Repository”.

When it’s imported go to the Java or Java EE perspective and import the project into your workspace. It will list it as two projects, one being the Maven one. Import both.

This is very similar in IntelliJ.

Because the project has not been created with a Spring Boot wizard you may have to do a few things manually.

You can rebuild Maven by right clicking on the pom.xml file and selecting Maven->Update Project

# Addendum

If you have been using Lombok then you can add it to your project in Maven with:

<dependencies>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version>

<scope>provided</scope>

</dependency>

</dependencies>

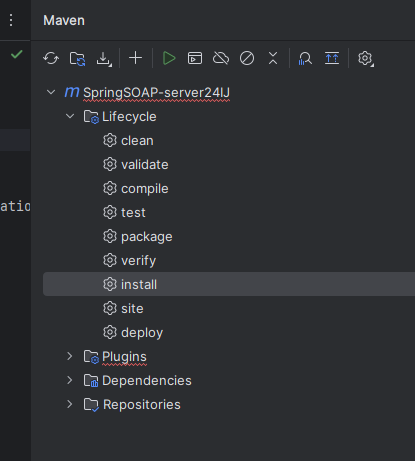
Link to GitHub with several working examples.

<https://github.com/LBU-SESC>

Problems in the lab using IntelliJ

The lab doesn’t allow maven archetypes to be downloaded and you get an error on the jaxb2 plugin and the MainApplication (SpringBootApplication class).

To get rid, go to view-tool window-maven



Right click on “install” and select “run maven build”.

The right click on pom.xml and reload maven project.

<https://brainly.com/question/43508642>

Non-resolvable parent POM for com.example:producingwebservice:0.0.1-SNAPSHOT: The following artifacts could not be resolved: org.springframework.boot:spring-boot-starter-parent:pom:3.2.2 (absent): org.springframework.boot:spring-boot-starter-parent:pom:3.2.2 failed to transfer from https://repo.maven.apache.org/maven2 during a previous attempt. This failure was cached in the local repository and resolution is not reattempted until the update interval of central has elapsed or updates are forced. Original error: Could not transfer artifact org.springframework.boot:spring-boot-starter-parent:pom:3.2.2 from/to central (https://repo.maven.apache.org/maven2): PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target and 'parent.relativePath' points at no local POM