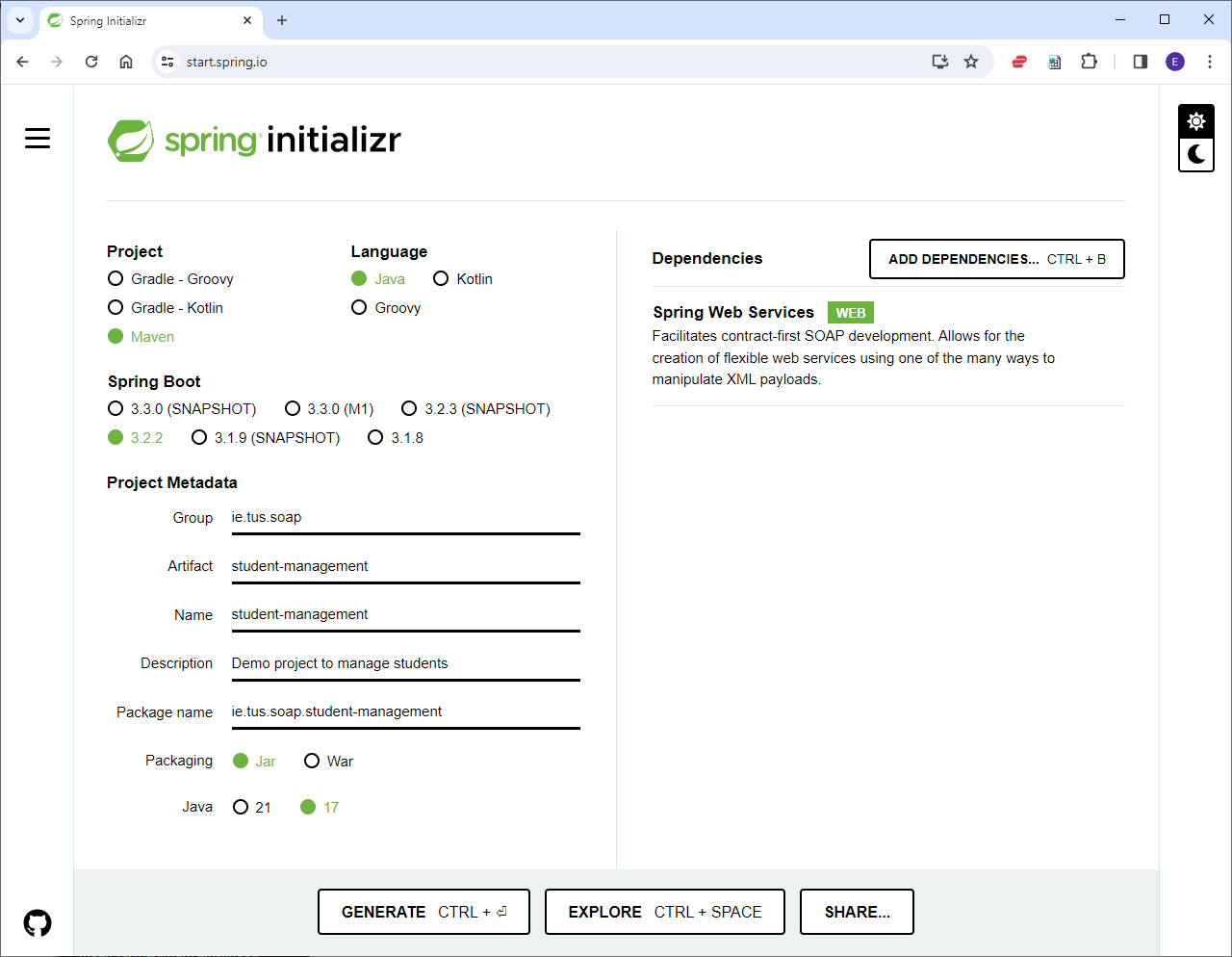
**SOAP 1 – Basic Service**

~~We’ll use Spring Boot for the SOAP and REST labs. To get a starter project go to ‘start.spring.io’ and fill out the form as follows:~~



~~Note:~~

* ~~The Spring Web Services dependency is needed for SOAP web services.~~
* ~~Check what version of java is installed on your machine before picking the version~~
* ~~Use whatever is latest version 3.x.x and not a snapshot)~~

~~Click ‘Generate’ to download the starter project.~~

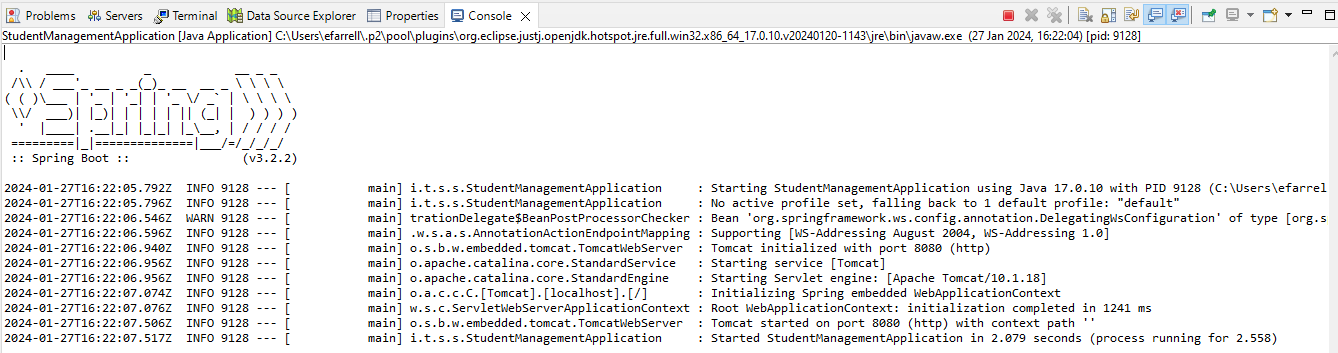
~~Extract the starter project to disk, and then in Eclipse -> ‘File – Import – Maven – Existing Maven Projects’. You should have the following in Eclipse:~~

Graphical user interface, text, application

Description automatically generated

~~To run an application in Spring Boot, right-click on the StudentManagementApplication.java -> Run-as – Java Application.~~

~~Note that Spring Boot comes with Apache Tomcat embedded so there’s no need to install a server like you did in 2~~~~nd~~ ~~year.~~



**Step 2: Create a SOAP service**

There are two ways to create a web service: Contract-Last and Contract-First. In the contract-last approach, the WSDL is generated from the Java code. In contract-first, we start with the WSDL contract, and use it to generate the Java classes. We’ll use contract-first.

We do this by creating xml files for the request and response. Then create an xsd file to define the structure of the request and response files. The xsd is used to generate the Java files.

~~Create a new folder called xml-files in the project, and then create the following request, response and xsd files:~~

~~The project should then look as follows (xml code on the next page):~~

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

**Step 3: JAXB**

~~JAXB is a package for converting from XML to Java classes and vice-versa. In our project we will have the following classes created from the xsd file:~~

* ~~GetStudentDetailsRequest~~
* ~~GetStudentDetailsResponse~~
* ~~StudentDetails (as its defined as a type in the xsd file)~~

1. ~~Copy the student-details.xsd to the ‘src/main/resources’ folder.~~
2. ~~Edit the pom.xml file to add the JAXB2 maven plugin (Add the following under plugins)~~

<!-- JAXB2 maven plugin -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>jaxb2-maven-plugin</artifactId>

<version>3.1.0</version>

<executions>

<execution>

<id>xjc</id>

<goals>

<goal>xjc</goal>

</goals>

</execution>

</executions>

<configuration>

<sources>

<source>${project.basedir}/src/main/resources/student-details.xsd

</source>

</sources>

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

<clearOutputDir>false</clearOutputDir>

</configuration>

</plugin>

~~(CTRL–SHIFT–F to autoformat(indent))~~

~~If you have an error at this stage, right-click on the project –> Maven – Update Project.~~

~~Notice that several files were automatically created by the JAXB plugin:~~

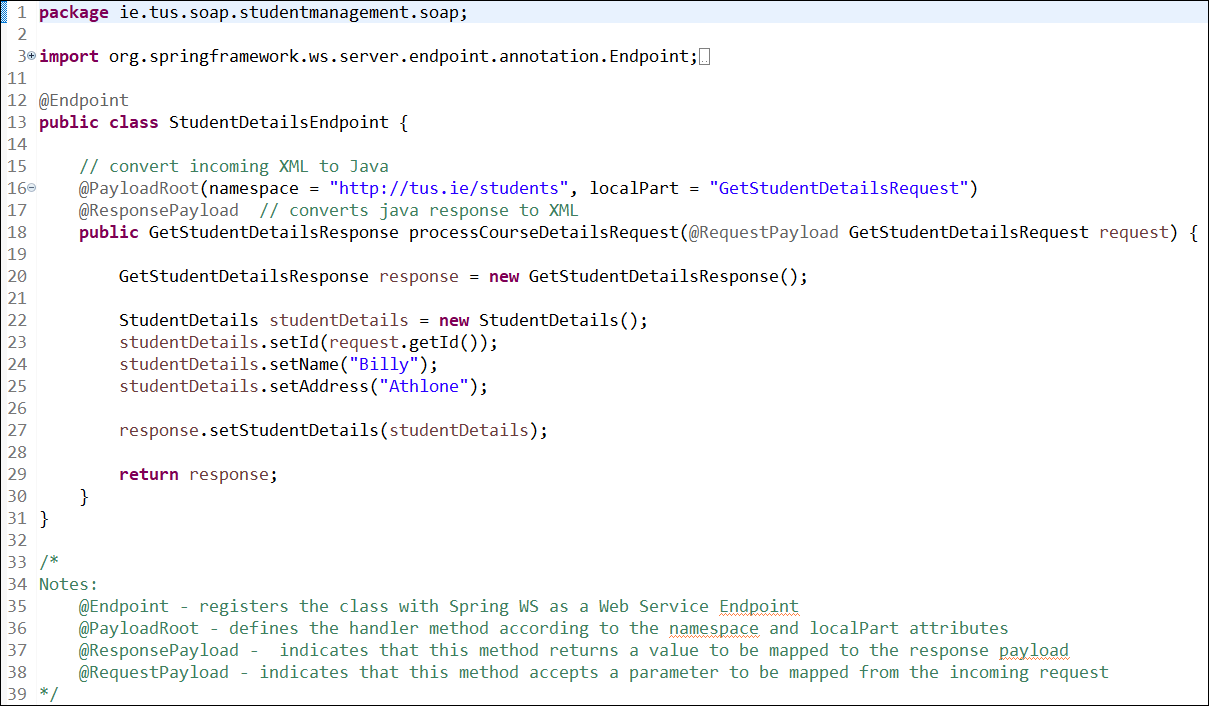
Graphical user interface, application

Description automatically generated with medium confidence

**~~Step 4: Create an Endpoint~~**

~~An endpoint class will handle incoming requests for the service and send back appropriate responses. It’s defined as follows:~~

~~Create a new Java Class called StudentDetailsEndpoint (Note the package)~~



~~Note: The above file will send back “Billy” and “Athlone” no matter what ID is requested. We’ll change this later.~~

**~~Step 5: Create a configuration class~~**

~~Create a new class called WebServiceConfig. Note the package.~~

~~This class is used to set up a servlet to handle incoming requests from ‘/ws/\*’~~



The @Bean annotation indicates that a method produces a bean(object) to be managed by the Spring container. So, in this case, the method will be run and the servlet will be registered for /ws/\*. i.e. requests for /ws/\* will be dispatched to the Endpoint.

The package structure of the project should now be as follows:

Graphical user interface, text, application

Description automatically generated with medium confidence

**~~Step 6: Generating WSDL~~**

~~We need to~~ **~~add the following methods to the WebServiceConfig~~** ~~from previous step. These are the methods that look after the generation of the WSDL file from the student-details.xsd.~~

~~Note that all of these are decorated with the @Bean annotation. If a @Bean is not given a name, it just takes the name of the method. Lines 36-39 create a bean called studentsSchema, and this is passed into the method on line 27 (managed by Spring).~~

Text

Description automatically generated

~~The last thing we need before we run the project is to add the following to the dependencies section in the pom.xml file.~~

<dependency>

<groupId>wsdl4j</groupId>

<artifactId>wsdl4j</artifactId>

</dependency>

**Step 7: Running the service**

We can run the service as before by right-clicking on StudentManagementApplication.java – Run as – Java Application (you may have to rebuild the project first).

In the browser, go to ‘localhost:8080/ws/students.wsdl’. This is the wsdl for the service.Text

Description automatically generated with medium confidence

**Step 8: Using Wizdler to execute SOAP requests**

~~Search Google for ‘wizdler plugin’ and install in Chrome.~~

~~With the plugin installed, refresh the page and you should be able to send a request:~~

Graphical user interface, text, application, email

Description automatically generated

Fill in a number for id (in place of [int]), and click ‘Go’. This sends the ‘GetStudentDetailsRequest’.

Graphical user interface, text, application, email

Description automatically generated

Observe the ‘GetStudentDetailsResponse’ response:

Graphical user interface, text, application, email

Description automatically generated

**Step 9: Review**

1. ~~Go back over the lab and review the steps.~~
2. Take a closer look at the WSDL file and compare with what we had in the lecture. Can you see the different parts (Types, Messages, Port Types, Bindings, Service Ports)?