Systemy rozproszone i integracja usług - ćwiczenia nr 4: implementacja usługi typu SOAP

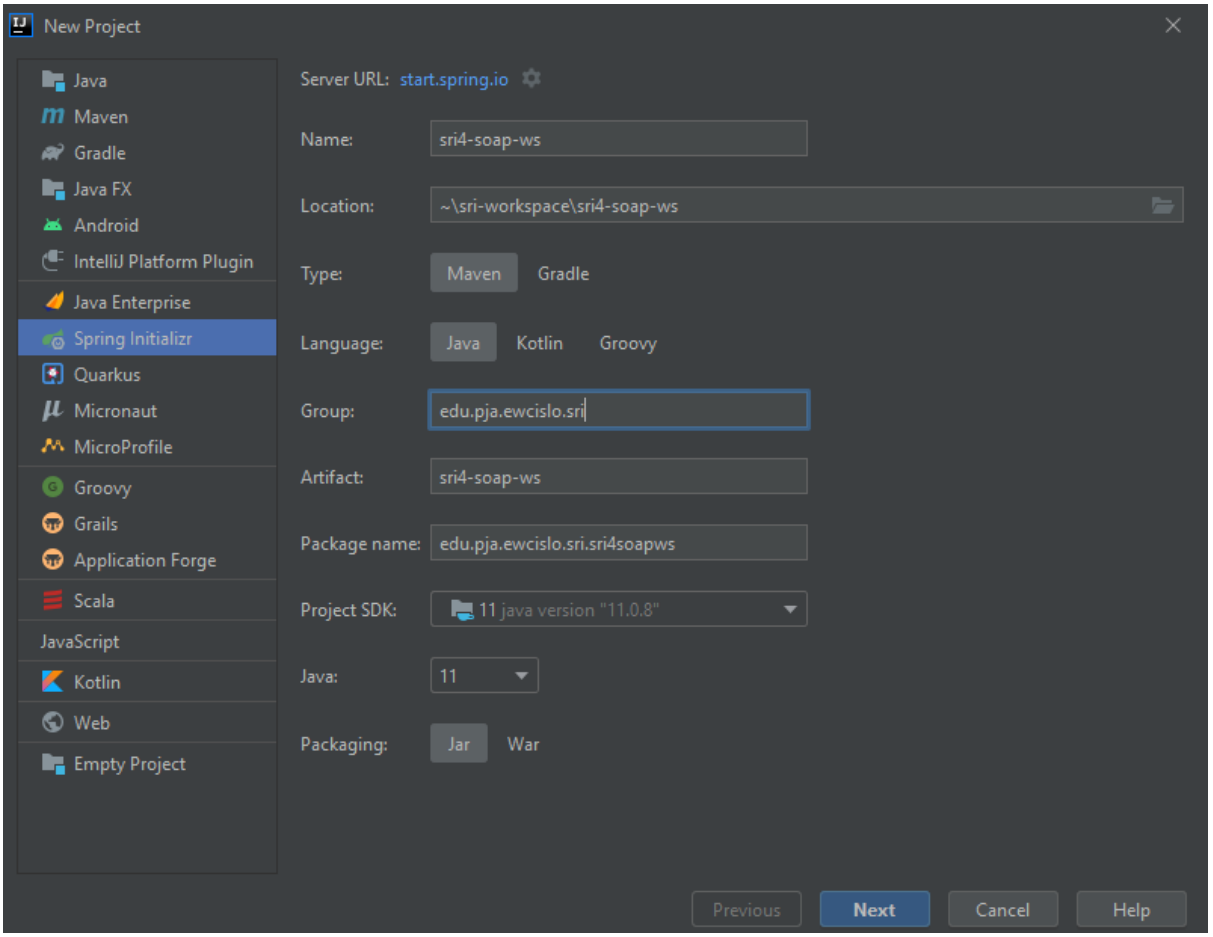
Celem ćwiczenia jest nabycie umiejętności implementacji usługi opartej na XML typu SOAP

Część 1: zadanie wprowadzające

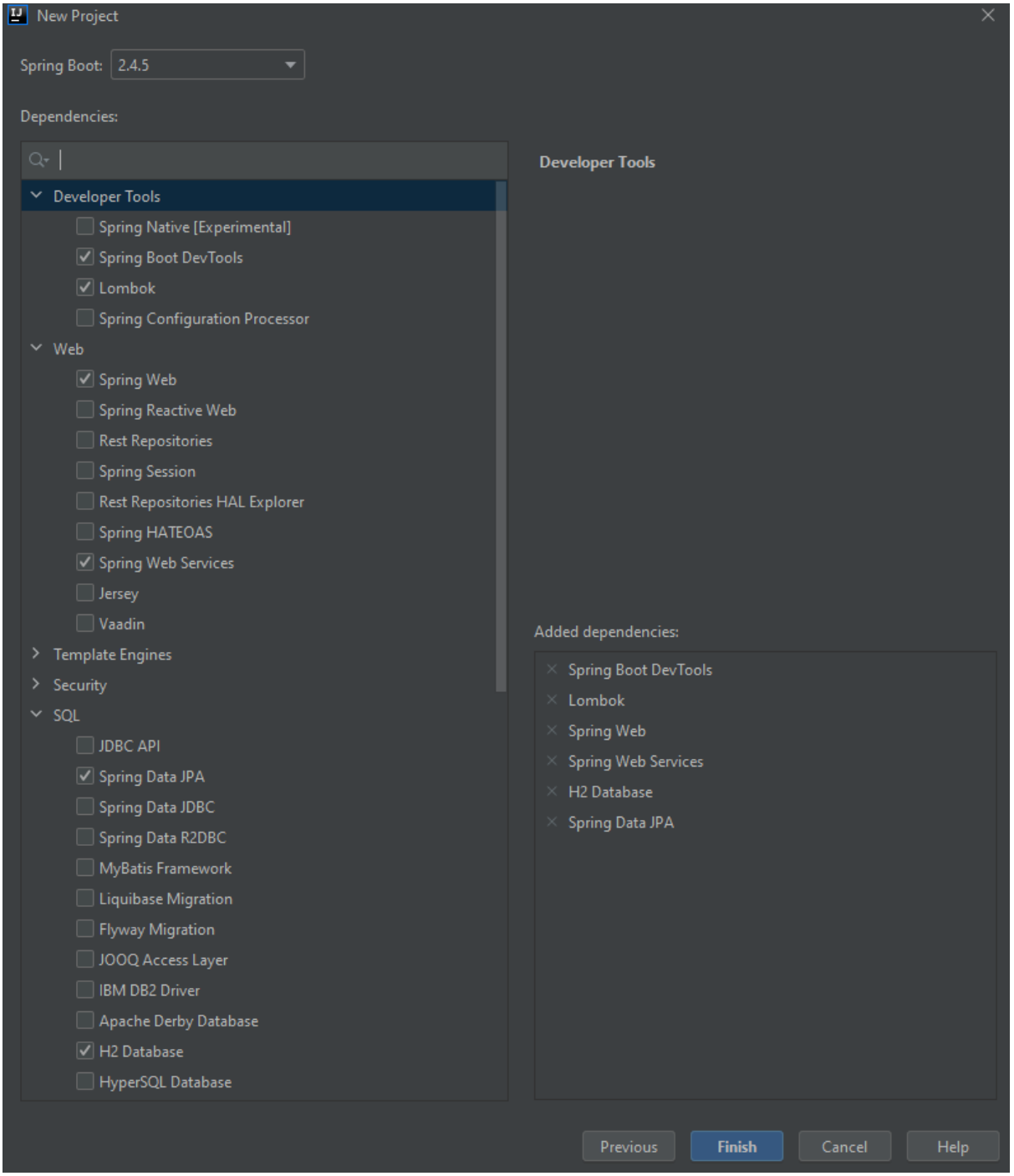
W ramach tego ćwiczenia zaimplementujemy przykładową usługę SOAP w technologii Java Spring. Zakres funkcjonalny będzie podobny do zadania 2 - będziemy udostępniać kilka podstawowych operacji na danych opartych na modelu z ćwiczenia 2.

1. Niezbędne oprogramowanie.
   Do wykonania tego ćwiczenia niezbędne będzie środowisko Java (rekomendowane JDK Oracle 11 LTS), oraz środowisko developerskie (rekomendowane IntelliJ Idea Ultimate). Instrukcje instalacji znajdują się w opisie zadania 2.
2. Tworzenie szkieletu aplikacji
   Podobnie jak w poprzednich ćwiczeniach wykorzystamy Spring Boot Initializer do tworzenia szkieletu aplikacji.

2.1. Tworzenie nowego projektu
   W programie IntelliJ Idea Ultimate należy wybrać opcję New -> Project… z menu File Następnie należy wybrać ‘Spring Initializr’ z menu po lewej stronie okna dialogowego i uzupełnić podstawowe dane projektu:



Po wybraniu ‘Next’ należy zdefiniować zależności projektu, jak na poniższym zrzucie ekranu:



2.2. Uzupełnienie zależności w pom.xml:

W sekcji ‘dependencies’ należy dodać następujący fragment:

```
<dependency>
  <groupId>wsdl4j</groupId>
  <artifactId>wsdl4j</artifactId>
</dependency>
```

A w sekcji ‘plugins’ należy dodać poniższy fragment:

```
<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/employees.xsd</source>
    </sources>
  </configuration>
</plugin>
```

Po zapisaniu zaktualizowanego pliku pom.xml należy kliknąć prawym klawiszem myszy na głównej gałęzi projektu w oknie po lewej stronie i z menu kontekstowego wybrać Maven -> Reload project. Nowe biblioteki powinny zostać pobrane i zainstalowane automatycznie.

3. Konfiguracja projektu
   Należy upewnić się, że JDK, procesor adnotacji i plugin Lombok zostały poprawnie skonfigurowane. Szczegółowy opis znajduje się w opisie poprzedniego zadania 2 w punktach 3.1 - 3.3
4. Konfiguracja w pliku application.properties
   Podobnie jak w zadaniu 2 wykorzystamy bazę danych H2 do przechowywania danych, na których będą wykonywane operacje wywoływane przez usługę sieciową.

   ```
   spring.datasource.url=jdbc:h2:mem:sri-hr
   spring.datasource.driverClassName=org.h2.Driver
   spring.datasource.username=sa
   spring.datasource.password=
   spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
   spring.h2.console.enabled=true
   ```

5. Tworzenie modelu i repozytorium
   Wykorzystamy klasy stworzone w ramach zadania 2:

Employee.java

```java
package edu.pja.sri.ewcislo.sri02.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.time.LocalDate;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String firstName;
    private String lastName;
    private LocalDate birthDate;
    private String job;
}
```

EmployeeRepository.java

```java
package edu.pja.sri.ewcislo.sri02.repo;

import edu.pja.sri.ewcislo.sri02.model.Employee;
import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface EmployeeRepository extends CrudRepository<Employee, Long> {
    List<Employee> findAll();
}
```

6. Tworzenie schematu XML komunikatów.
Należy utworzyć plik employees.xsd w katalogu src/main/resources. Będzie zawierać on definicję typów danych używanych w interfejsie usługi sieciowej, oraz definicję komunikatów:

employees.xsd

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://sri4soapws.ewcislo.sri.pja.edu/employees"
           targetNamespace="http://sri4soapws.ewcislo.sri.pja.edu/employees"
           elementFormDefault="qualified">

    <xs:complexType name="employeeDto">
        <xs:sequence>
            <xs:element name="id" type="xs:decimal" minOccurs="0"/>
            <xs:element name="firstName" type="xs:string"/>
            <xs:element name="lastName" type="xs:string"/>
            <xs:element name="birthDate" type="xs:date"/>
            <xs:element name="job" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="getEmployeesRequest">
        <xs:complexType>
        </xs:complexType>
    </xs:element>

    <xs:element name="getEmployeesResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="employees" type="tns:employeeDto" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getEmployeeByIdRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="employeeId" type="xs:decimal"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="getEmployeeByIdResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="employee" type="tns:employeeDto" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="addEmployeeRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="employee" type="tns:employeeDto"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="addEmployeeResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="employeeId" type="xs:decimal" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

W naszym projekcie będziemy tworzyć usługę z trzema metodami: getEmployees(), getEmployeeById(id) i addEmployee(employee). Dla każdej z tych metod trzeba zdefiniować komunikat wejściowy i wyjściowy. Należy opisać również wszystkie własne typy danych używane w komunikacji. W naszym przykładzie opisana została klasa Employee. Więcej na temat definiowania typów danych w standardzie XML Schema można znaleźć tutaj: https://www.w3schools.com/xml/schema_intro.asp

7. Konfiguracja usługi SOAP
   Należy utworzyć klasę SoapWSConfig zawierającą następującą treść:

SoapWSConfig.java

```java
package edu.pja.ewcislo.sri.sri4soapws.config;

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 SoapWSConfig extends WsConfigurerAdapter {

    public static final String EMPLOYEE_NAMESPACE = "http://sri4soapws.ewcislo.sri.pja.edu/employees";

    @Bean
    public ServletRegistrationBean<MessageDispatcherServlet> messageDispatcherServlet(ApplicationContext applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean<>(servlet, urlMappings: "/ws/*");
    }

    @Bean(name = "employees")
    public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema employeesSchema) {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("EmployeesPort");
        wsdl11Definition.setLocationUri("/ws/employees");
        wsdl11Definition.setTargetNamespace(EMPLOYEE_NAMESPACE);
        wsdl11Definition.setSchema(employeesSchema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema countriesSchema() { return new SimpleXsdSchema(new ClassPathResource("employees.xsd")); }
}
```

Zawiera ona podstawową konfigurację usługi i podłączenie jej do stworzonego wcześniej schematu danych w pliku xsd.

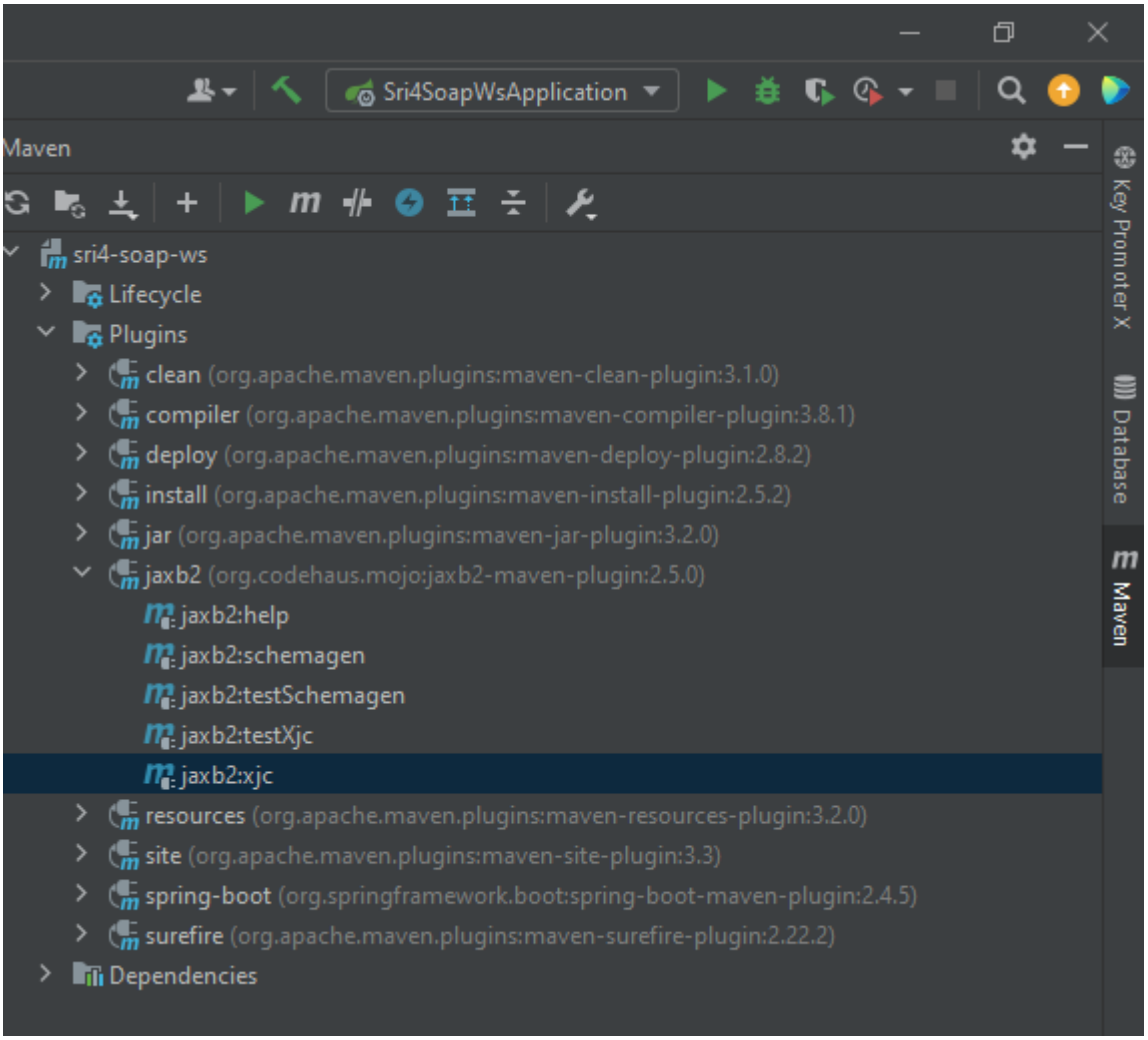
8. Tworzenie danych początkowych
Opcjonalnie można stworzyć klasę DataInitializer, która będzie odpowiedzialna za zasilenie bazy danych przy starcie aplikacji:

DataInitializer.java

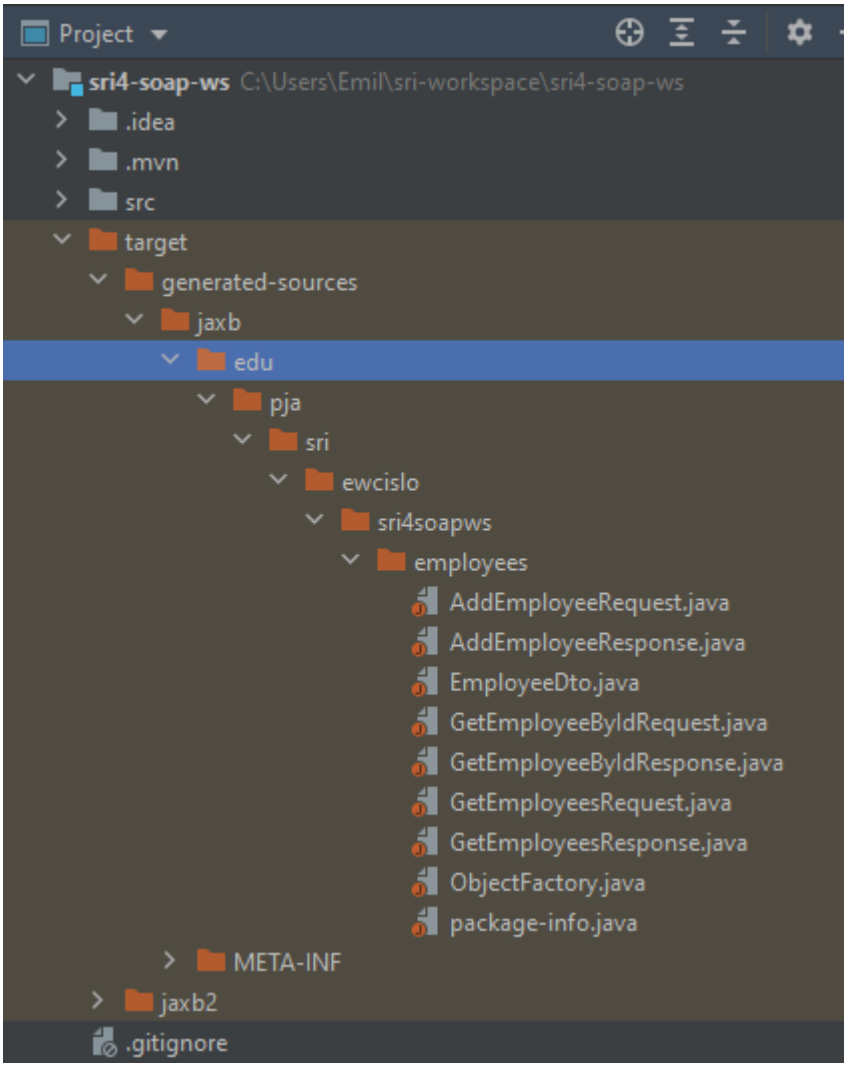
```java
package edu.pja.ewcislo.sri.sri4soapws;

import edu.pja.ewcislo.sri.sri4soapws.model.Employee;
import edu.pja.ewcislo.sri.sri4soapws.repo.EmployeeRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.stereotype.Component;

import java.time.LocalDate;
import java.util.Arrays;

@Component
@RequiredArgsConstructor
public class DataInitializer implements ApplicationListener<ContextRefreshedEvent> {

    private static final Logger LOG = LoggerFactory.getLogger(DataInitializer.class);

    private final EmployeeRepository employeeRepository;

    public void initData() {
        Employee e1 = Employee.builder()
                .firstName("Jan")
                .lastName("Kowalski")
                .job("Clerk")
                .birthDate(LocalDate.of( year: 1990,  month: 01,  dayOfMonth: 01))
                .build();
        Employee e2 = Employee.builder()
                .firstName("Adam")
                .lastName("Nowak")
                .job("Manager")
                .birthDate(LocalDate.of( year: 1991,  month: 01,  dayOfMonth: 01))
                .build();
        Employee e3 = Employee.builder()
                .firstName("Anna")
                .lastName("Iksińska")
                .job("Assistant")
                .birthDate(LocalDate.of( year: 1992,  month: 01,  dayOfMonth: 01))
                .build();
        employeeRepository.saveAll(Arrays.asList(e1, e2, e3));
        LOG.info("Data initialized");
    }

    @Override
    public void onApplicationEvent(ContextRefreshedEvent event) { initData(); }
}
```

9. Generowanie klas na podstawie schematu xsd

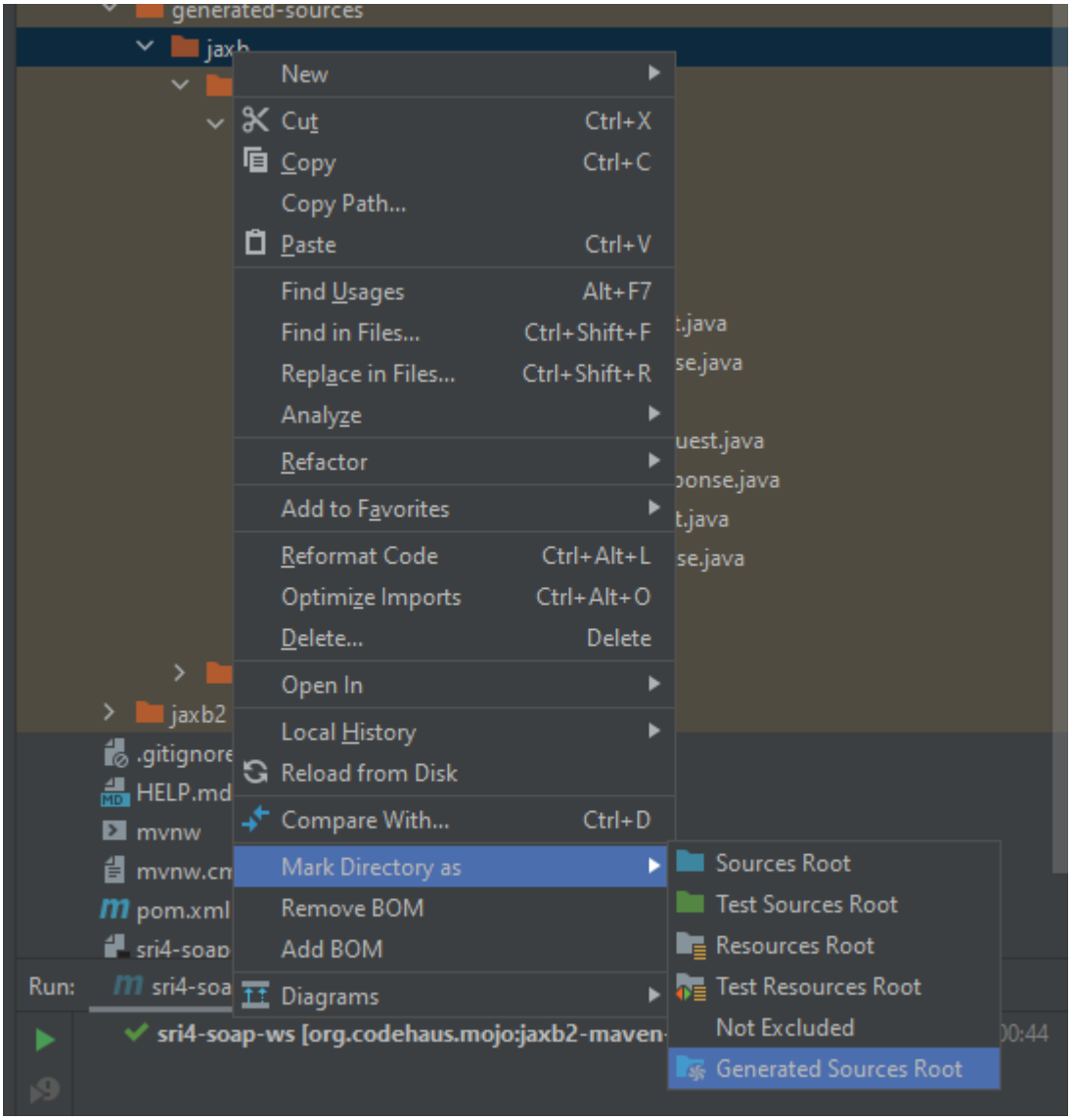
Ponieważ usługa typu SOAP korzysta z modelu danych opisanych w schemacie XML Schema, potrzebujemy klas wygenerowanych na podstawie tego schematu, aby wykorzystać je w implementacji usługi. W tym celu uruchamiamy plugin jaxb2 dodany wcześniej do pliku pom.xml. Aby to zrobić, należy otworzyć zakładkę ‘Maven’ w środowisku IntelliJ (domyślnie przy prawej krawędzi) i wybrać zadanie ‘jaxb2:xjc’ z katalogu ‘Plugins’:



Po wykonaniu zadania w katalogu target powinny pojawić się wygenerowane klasy:



Klas tych na razie nie możemy wykorzystać w projekcie (nie są dodane do classpath projektu). Aby to naprawić, należy kliknąć prawym klawiszem myszy na folderze jaxb i wybrać opcję ‘Mark Directory as Generated Sources Root’:



10. Tworzenie implementacji usługi
W tym celu należy stworzyć klasę EmployeeEndpoint o następującej treści:

EmployeeEndpoint.java

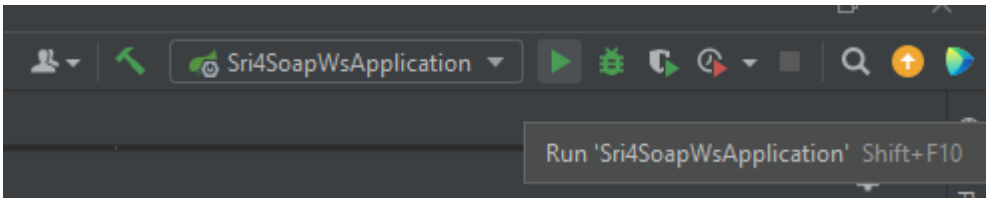
```java
package edu.pja.ewcislo.sri.sri4soapws.endpoint;

import edu.pja.ewcislo.sri.sri4soapws.config.SoapWSConfig;
import edu.pja.ewcislo.sri.sri4soapws.model.Employee;
import edu.pja.ewcislo.sri.sri4soapws.repo.EmployeeRepository;
import edu.pja.sri.ewcislo.sri4soapws.employees.*;
import lombok.RequiredArgsConstructor;
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 javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;
import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Endpoint
@RequiredArgsConstructor
public class EmployeeEndpoint {

    private final EmployeeRepository employeeRepository;

    @PayloadRoot(namespace = SoapWSConfig.EMPLOYEE_NAMESPACE, localPart = "getEmployeesRequest")
    @ResponsePayload
    public GetEmployeesResponse getEmployees(@RequestPayload GetEmployeesRequest req) {
        List<Employee> employeeList = employeeRepository.findAll();
        List<EmployeeDto> dtoList = employeeList.stream()
                .map( this::convertToDto )
                .collect(Collectors.toList());
        GetEmployeesResponse res = new GetEmployeesResponse();
        res.getEmployees().addAll(dtoList);
        return res;
    }

    @PayloadRoot(namespace = SoapWSConfig.EMPLOYEE_NAMESPACE, localPart = "getEmployeeByIdRequest")
    @ResponsePayload
    public GetEmployeeByIdResponse getEmployeeById(@RequestPayload GetEmployeeByIdRequest req) {
        Long employeeId = req.getEmployeeId().longValue();
        Optional<Employee> emp = employeeRepository.findById(employeeId);
        GetEmployeeByIdResponse res = new GetEmployeeByIdResponse();
        res.setEmployee(convertToDto(emp.orElse( other: null)));
        return res;
    }

    @PayloadRoot(namespace = SoapWSConfig.EMPLOYEE_NAMESPACE, localPart = "addEmployeeRequest")
    @ResponsePayload
    public AddEmployeeResponse addEmployee(@RequestPayload AddEmployeeRequest req) {
        EmployeeDto empDto = req.getEmployee();
        Employee emp = convertToEntity(empDto);
        employeeRepository.save(emp);
        AddEmployeeResponse response = new AddEmployeeResponse();
        response.setEmployeeId(new BigDecimal(emp.getId()));
        return response;
    }
```

```java
    private EmployeeDto convertToDto(Employee e) {
        if(e == null) {
            return null;
        }
        try {
            EmployeeDto dto = new EmployeeDto();
            dto.setId(new BigDecimal(e.getId()));
            dto.setFirstName(e.getFirstName());
            dto.setLastName(e.getLastName());
            XMLGregorianCalendar birthDate = null;
            birthDate = DatatypeFactory.newInstance().newXMLGregorianCalendar(e.getBirthDate().toString());
            dto.setBirthDate(birthDate);
            dto.setJob(e.getJob());

            return dto;
        } catch (DatatypeConfigurationException datatypeConfigurationException) {
            datatypeConfigurationException.printStackTrace();
            return null;
        }
    }

    private Employee convertToEntity(EmployeeDto dto) {
        return Employee.builder()
                .id(dto.getId() != null ? dto.getId().longValue() : null)
                .firstName(dto.getFirstName())
                .lastName(dto.getLastName())
                .job(dto.getJob())
                .build();
    }

}
```

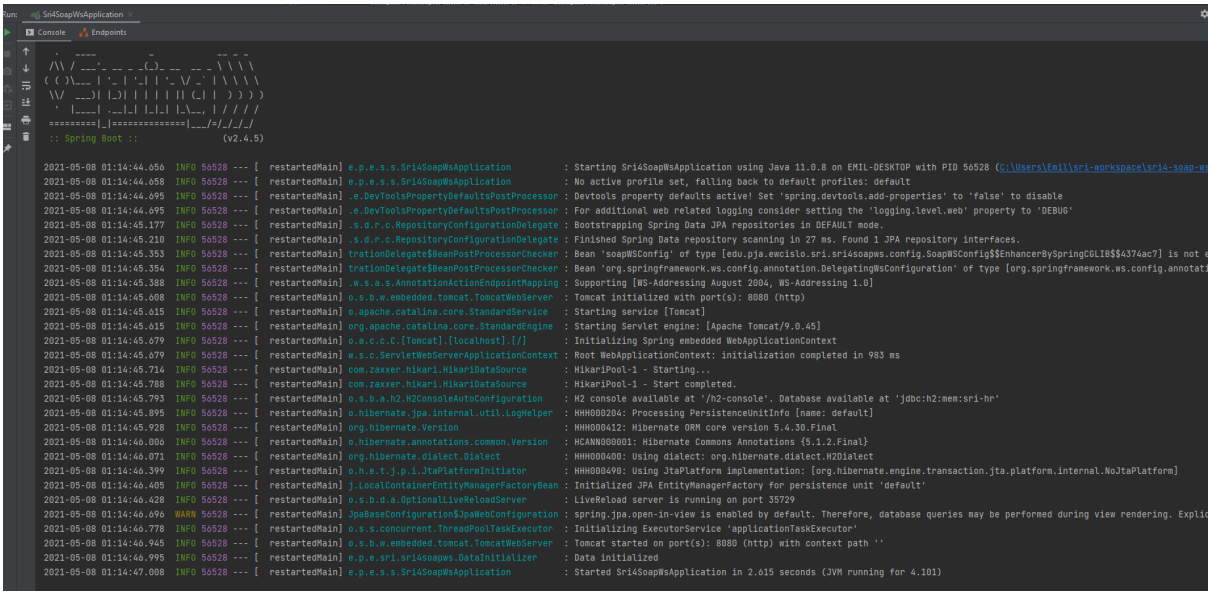
Nasza usługa zawiera trzy metody biznesowe: getEmployees(), getEmployeeById() i addEmployee(). Jako parametry tych metod oraz rezultat wykorzystujemy klasy wygenerowane na podstawie schematu xsd. Ponieważ konieczna jest konwersja z wygenerowanej klasy EmployeeDto na klasę encji Employee i odwrotnie, stworzymy metody konwertujące: convertToDto() i convertToEntity().

11. Uruchomienie aplikacji

Aplikację uruchamiamy klikając na zieloną strzałkę w prawym górnym rogu ekranu, lub za pomocą skrótu klawiszowego Shift-F10:



Po uruchomieniu na konsoli nie powinny pojawić się błędy:



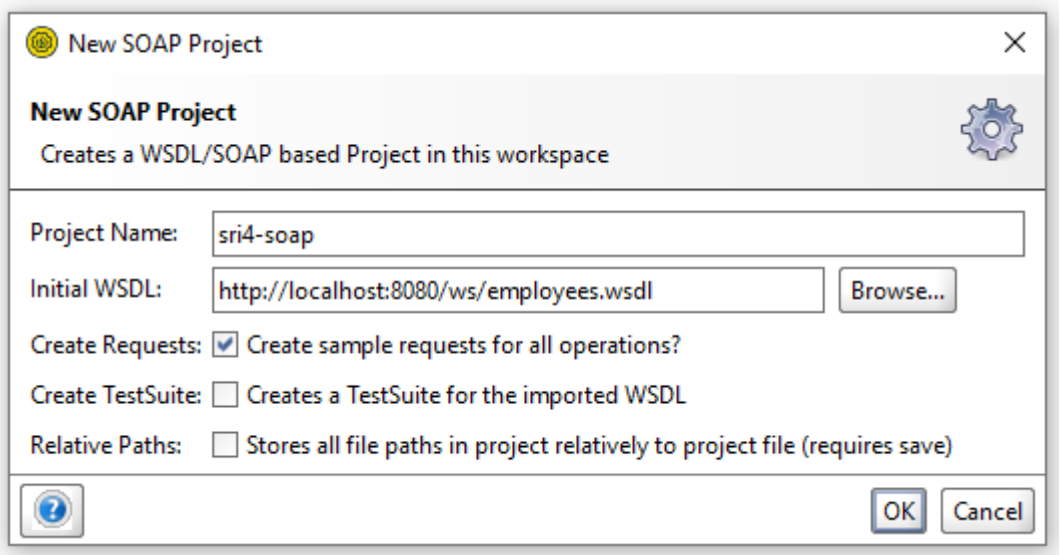
12. Testowanie usługi SOAP za pomocą programu SoapUI

12.1. Instalacja oprogramowania

Ze strony https://www.soapui.org/downloads/soapui/ należy pobrać SoapUI w wersji open source i zainstalować.

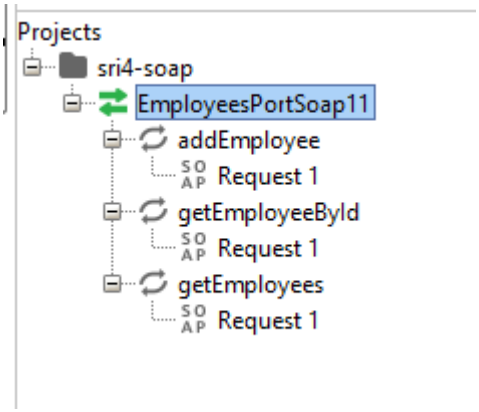
12.2. Testowanie usługi

Po uruchomieniu programu SoapUI należy stworzyć nowy projekt wybierając z menu File -> New SOAP Project:

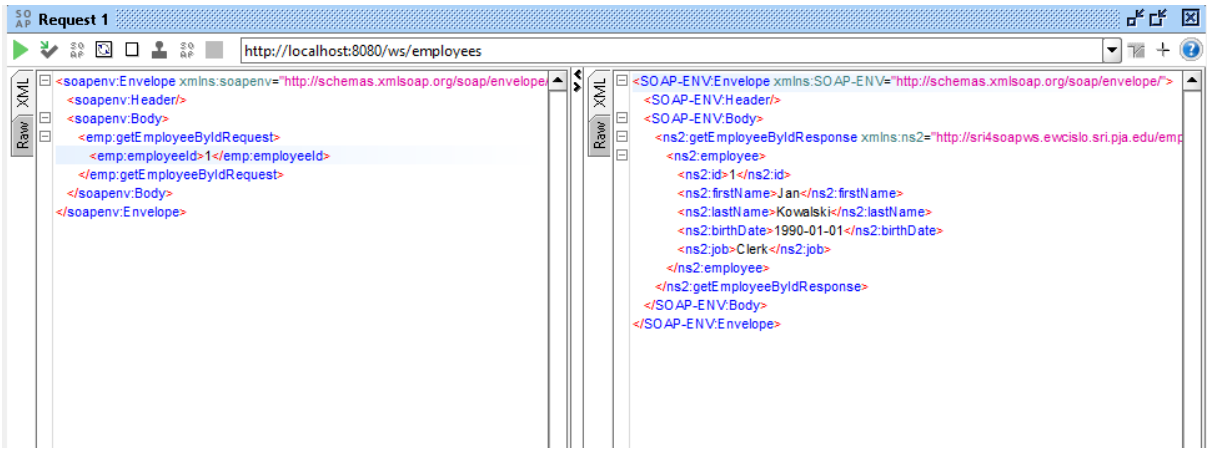


Należy uzupełnić nazwę projektu oraz wskazanie na lokalizację pliku WSDL zawierający opis usługi. Opis ten zostanie wygenerowany automatycznie przez naszą aplikację po jej uruchomieniu. Publikowany jest pod adresem: http://localhost:8080/ws/employees.wsdl

Po utworzeniu projektu SoapUI odczyta schemat usługi i wygeneruje przykładowe żądania:



Po wybraniu żądania i uzupełnieniu danych można przetestować daną metodę klikając na zieloną strzałkę uruchamiającą żądanie:



Część 2: zadanie na ocenę

Zaprojektuj i zaimplementuj system realizujący dowolną funkcjonalność biznesową, udostępniony za pomocą usługi typu SOAP. Minimalne wymagania to:

1. Usługa musi udostępniać co najmniej 3 metody biznesowe.
2. Należy zastosować własne klasy do jako parametry metod oraz jako typy danych zwracanych przez metody.
3. Projekt musi mieć sensowną funkcjonalność biznesową.

Projekt należy przetestować za pomocą SoapUI, lub podobnego narzędzia.
Gotowy projekt i udokumentowane testy (np. w postaci zrzutów ekranu z narzędzia testującego) należy wysłać jako rozwiązanie.