SOAP

→ Project

Start.spring.io → Dependencies

- Web Services
- JPA
- H2 → In-memory database.

- Make a new folder example.
  - → Create Request.xml

*Untitled 2 | Request.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <GetCourseDetailsRequest xmlns="http://in28minutes.com/courses">
3     <id>123</id>
4 </GetCourseDetailsRequest>
```

→ response.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<GetCourseDetailsResponse xmlns="http://in28minutes.com/courses">
    <CourseDetails>
        <id>123</id>
        <name>Spring in28minutes</name>
        <description>You would learn the basic</description>
    </CourseDetails>
</GetCourseDetailsResponse>
```

→ In example folder, create xsd file, course-details.xsd.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://in28minutes.com/courses"
xmlns:tns="http://in28minutes.com/courses" elementFormDefault="qualified">
    <element name="GetCourseDetailsRequest">
        <complexType>
            <sequence>
                <element name= "id" type="integer"></element>
            </sequence>
        </complexType>
    </element>
</schema>
```

→ Now, link request.xml & xsd together

→ Show the validation, by performing errors.

→ Now in xsd, copy GetCourseDetailsRequest & use for CourseDetailsResponse. Also, we need to define returntype.

```xml
<element name="GetCourseDetailsResponse">
    <complexType>
        <sequence>
            <element name= "CourseDetails" type="tns:CourseDetails"></eleme
        </sequence>
    </complexType>
</element>

<complexType name="CourseDetails">
    <sequence>
        <element name="id" type="integer"/>
        <element name="name" type="string"/>
        <element name="description" type="string"/>
    </sequence>
</complexType>
```

→ Now add schema to response.xml.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<GetCourseDetailsResponse xmlns="http://in28minutes.com/courses"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://in28minutes.com/courses course-details.xsd">
    <CourseDetails>
        <id>123</id>
        <name>Spring in28minutes</name>
        <description>You would learn the basics of Spring Framework</descriptio
    </CourseDetails>
</GetCourseDetailsResponse>
```

→ Give a prefix as xs to all the elements in my xsd.

Change xs: schema & then all the tags will change.

→ We defined XSD, now generate JAVA responses using JAXB.

- First, copy course-detail.xsd into resources for jaxb use.
- In pom.xml, add jaxb2-maven plugin; copy from ppt.
- After saving, a lot of classes will get generated in main/java.
- Change integer to int in xsd.

- Now create a class for endpoint, CourseDetailEndpoint

takes input - request.

output - response

```java
@Endpoint
public class CourseDetailsEndpoint {

    //method
    //input - GetCourseDetailsRequest
    //output - GetCourseDetailsResponse

    //http://in28minutes.com/courses
    //GetCourseDetailsRequest

    public GetCourseDetailsResponse
        processCourseDetailsRequest(GetCourseDetailsRequest request){

        GetCourseDetailsResponse response = new GetCourseDetailsResponse();

        return response;
```

→ Now, we need to do xml → java & java → xml conversions for req/res. So, add the annotation & lets configure the response.

```java
    // http://in28minutes.com/courses
    // GetCourseDetailsRequest
    @PayloadRoot(namespace = "http://in28minutes.com/courses", localPart = "Get
    @ResponsePayload
    public GetCourseDetailsResponse processCourseDetailsRequest(@RequestPayload
        GetCourseDetailsResponse response = new GetCourseDetailsResponse();

        CourseDetails courseDetails = new CourseDetails();
        courseDetails.setId(request.getId());
        courseDetails.setName("Microservices Course");
        courseDetails.setDescription("That would be a wonderful course!");

        response.setCourseDetails(courseDetails);

        return response;
    }

}
```

-) Now before creating the endpoint url, we need to create the WSDL contract.

Now we initialize spring web services, make a class,

WebServiceConfig.

- MessageDispatcherServlet will help to configure endpoints for a SOAP service.
- ServletRegistrationBean will help match a servlet to a url.
- Servlet → is any program that helps configure request-response.

*WebServiceConfig.java

```java
//Enable Spring Web Services
@EnableWs
// Spring Configuration
@Configuration
public class WebServiceConfig {
    // MessageDispatcherServlet
    // ApplicationContext
    // url -> /ws/*

    @Bean
    public ServletRegistrationBean messageDispatcherServlet(ApplicationContext context
        MessageDispatcherServlet messageDispatcherServlet =
                new MessageDispatcherServlet();
        messageDispatcherServlet.setApplicationContext(context);
        messageDispatcherServlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean(messageDispatcherServlet, "/ws/*");
    }
}
```

We have just defined a servlet and map a URL. That's it.

→ Let's configure a WSDL!!!

So spring uses xsd to generate WSDL. Let's get our schema first.

- XSD Schema is a return type in xml.

```java
@Bean
public XsdSchema coursesSchema(){
    return new SimpleXsdSchema(new ClassPathResource("course-details.xsd"));
}
```

→ Method to generate wsdl.

```
// /ws/courses.wsdl
// course-details.xsd
@Bean(name = "courses")
public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema coursesSch
    DefaultWsdl11Definition definition = new DefaultWsdl11Definition();
    definition.setPortTypeName("CoursePort");
    definition.setTargetNamespace("http://in28minutes.com/courses");
    definition.setLocationUri("/ws");
    definition.setSchema(coursesSchema);
    return definition;
}

@Bean
```

→ Add dependency in pom.xml

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

→ Use wizdler to test api.

→ open the api & send a request to get response.

localhost:8080/ws/courses.wsdl

→ The data is hard coded up until now, let's see how to retrieve that. For that, we will make a Bean, i.e. a new class name Course inside a package bean.

```
1  package com.in28minutes.soap.webservices.soapcoursemanagement.soap.bean;
2
3  public class Course {
4      private int id;
5      private String name;
6      private String description;
7
8
9
```

Generate getters and setters, constructor & toString.

→ Now we need to define a service that uses this Course, but first autowire that in our endpoint.

```java
@Endpoint
public class CourseDetailsEndpoint {

    @Autowired
    CourseDetailsService service;
```

→ Now create a class in a package service.

```java
@Component
public class CourseDetailsService {

    private static List<Course> courses = new ArrayList<>();

    //course - 1
    //Course findById(int id)

    //courses
    //List<Course> findAll()

    //deletecourse
    //Course deleteById(int id)

    //updating course & new course
}
```

Now we will define all these functions, but first let's have some dummy data to play.

```
private static List<Course> courses = new ArrayList<>();

static {
    Course course1 = new Course(1, "Spring", "10 Steps");
    courses.add(course1);

    Course course2 = new Course(2, "Spring MVC", "10 Examples");
    courses.add(course2);

    Course course3 = new Course(3, "Spring Boot", "6K Students");
    courses.add(course3);

    Course course4 = new Course(4, "Maven", "Most popular maven course on i
    courses.add(course4);
}
```

```
public Course findById(int id) {
    for (Course course : courses) {
        if (course.getId() == id)
            return course;
    }
    return null;
}

// courses
public List<Course> findAll() {
    return courses;
}
```

```
public int deleteById(int id) {
    Iterator<Course> iterator = courses.iterator();
    while (iterator.hasNext()) {
        Course course = iterator.next();
        if (course.getId() == id){
            iterator.remove();
            return 1;
        }
    }
    return 0;
}
```

→ Now, in Endpoints file, we add the service.

```
public GetCourseDetailsResponse processCourseDetailsRequest(@RequestPayloa

    GetCourseDetailsResponse response = new GetCourseDetailsResponse();

    Course course = service.findById(request.getId());

    CourseDetails courseDetails = new CourseDetails();

    courseDetails.setId(course.getId());

    courseDetails.setName(course.getName());

    courseDetails.setDescription(course.getDescription());

    response.setCourseDetails(courseDetails);
```

→ Use constructor to test passing the ID.

→ Refactoring to make a mapper.

```
@ResponsePayload
public GetCourseDetailsResponse processCourseDetailsRequest(@RequestPa

    Course course = service.findById(request.getId());

    return mapCourse(course);
}
```

```
private GetCourseDetailsResponse mapCourse(Course course) {
    GetCourseDetailsResponse response = new GetCourseDetailsResponse();

    CourseDetails courseDetails = new CourseDetails();

    courseDetails.setId(course.getId());

    courseDetails.setName(course.getName());

    courseDetails.setDescription(course.getDescription());

    response.setCourseDetails(courseDetails);

    return response;
}
```