



Escuela de Ingeniería Informática

Escuela de Ingeniería Informática
School of Computer Science Engineering

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Sistemas Distribuidos e Internet

Tema 10 Servicios web SOAP

SOAP
Contract First Web Services

RESTful API
GET PUT POST DELETE

Dr. Edward Rolando Núñez Valdez

nunezedward@uniovi.es

Índice

- Servicios web SOAP
 - ¿Qué son servicios web SOAP?
 - Arquitectura SOAP
 - Tecnologías de un servicio web SOAP
 - Protocolo de comunicación SOAP
 - Sobre SOAP (envelope)
 - WSDL - Autodescripción de los servicios web SOAP
 - Ventajas y desventajas de SOAP
 - SOAP vs REST
 - Implementación de SW SOAP con Spring Boot

¿Qué son servicios web SOAP?

- Un servicio web SOAP es un mecanismo para permitir la ***comunicación entre sistemas distribuidos*** a través de una red utilizando el ***protocolo de comunicación SOAP*** y el ***lenguaje XML como formato de datos***.
- Servicios web basados en protocolos abiertos y estándares.
- SOAP es el acrónimo de (***Simple Object Access Protocol***) o protocolo simple de acceso a objetos.
- SOAP es una especificación estandarizada y mantenida por el W3C.

Arquitectura de un servicio web SOAP

- **Proveedor del Servicio o service provider**

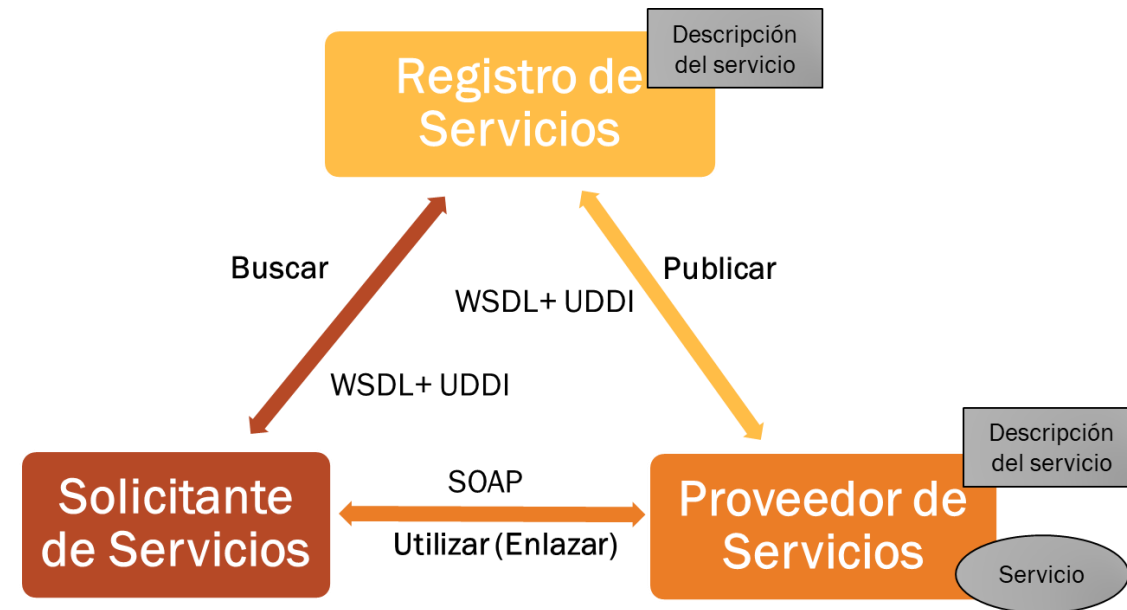
- Creador del servicio web que envía al publicador del servicio un fichero WSDL con la definición del servicio web.

- **Publicador de servicio o service registry**

- Proporciona el modo de descubrir tipos y semánticas de servicios web.
- Proporciona un enlace a dicho servicio.

- **Solicitante de servicio o service requester**

- Cliente del servicio web que contactará con el publicador para localizar servicios web.



Tecnologías de un servicio web SOAP

- SOAP (Simple Object Access Protocol)
 - **Protocolo estándar basado en XML** para el intercambio de información entre aplicaciones en entornos **descentralizados y distribuidos**.
- WSDL (Web Services Description Language)
 - Es un **lenguaje** basado en XML utilizado para **describir la funcionalidad** que proporciona un servicio Web.
- UDDI (Universal Description, Discovery and Integration)
 - **Registro público** pensado para **localizar** servicios web.
- XML (eXtensible Markup Language)
 - Estándar utilizado para **representar los datos** del servicio web
- HTTP (HyperText Transfer Protocol)
 - Protocolo del nivel de aplicación para la transferencia de mensajes

Mensajes	SOAP
Descripción	WSDL
Directorio	UDDI
Formato de datos	XML
Comunicación	HTTP, SMTP, ...

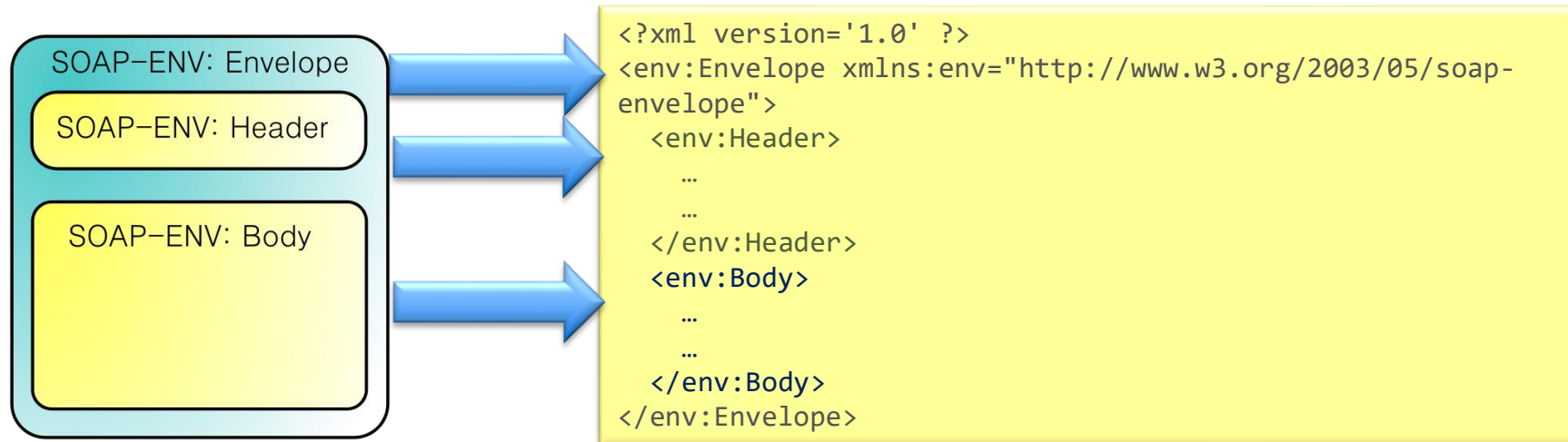
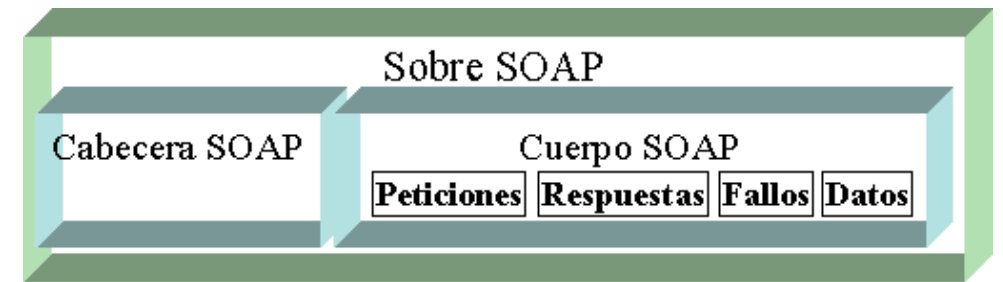
PROTOCOLO DE COMUNICACIÓN SOAP

Protocolo de comunicación - SOAP

- SOAP es un protocolo estándar **basado en XML** para el intercambio de información entre aplicaciones en entornos **descentralizados y distribuidos**.
 - La implementación más común de SOAP es sobre HTTP, aunque no es la única (SMTP, JMS, Otros.).
- Creado inicialmente por Microsoft, IBM y otros.
- Mantenido y actualizado por el consorcio del W3C.
- SOAP define una **estructura XML para los mensajes** intercambiados entre las aplicaciones.
- El mensaje es transportado en un sobre (envelope).

Sobre SOAP (envelope)

- El sobre o envelope (envoltorio) es el contenedor del mensaje SOAP -> **obligatorio**
- Tiene dos secciones
 - Encabezado (header) -> **opcional**
 - Cuerpo (body) -> **obligatoria**



Cabecera SOAP (Header)

- Contiene los *atributos opcionales* del mensaje.
- Elemento opcional útil para incluir meta-información sobre el mensaje SOAP.
- Permite extender un mensaje SOAP de forma modular y descentralizada (WS-Transaction, WS-Security, WS-Policy).

```
<env:Header>
  <m:reserva xmlns:m="[[http://www.mouta.com.ar]]"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <m:referencia>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:referencia>
    <m:fechaYHora>2001-11-29T13:20:00.000-05:00</m:fechaYHora>
  </m:reserva>
  <n:pasajero xmlns:n="http://miempresa.example.com/empleados"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <n:nombre>Áke Jógvan Øyvind</n:nombre>
  </n:pasajero >
</env:Header>
```

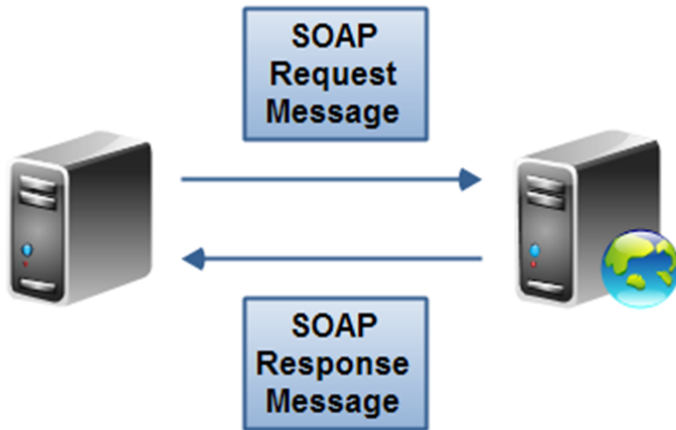
Cuerpo SOAP (Body)

- El cuerpo contiene la información relativa a la llamada/respuesta
 - Lógica de negocio
- Datos enviados como respuesta tras la ejecución de servicio.
- Información de error y estado.

```
<env:Body>
  <p:itinerario xmlns:p="http://empresaviajes.example.org/reserva/viaje">
    <p:ida>
      <p:salida>Nueva York</p:salida>
      <p:llegada>Los Angeles</p:llegada>
      <p:fechaSalida>2001-12-14</p:fechaSalida>
      <p:horaSalida>última hora de la tarde</p:horaSalida>
      <p:preferenciaAsiento>pasillo</p:preferenciaAsiento>
    </p:ida>
    <p:vuelta>
      <p:salida>Los Angeles</p:salida>
      <p:llegada>Nueva York</p:llegada>
      <p:fechaSalida>2001-12-20</p:fechaSalida>
      <p:horaSalida>media-mañana</p:horaSalida>
      <p:preferenciaAsiento />
    </p:vuelta>
  </p:itinerario>
  <q:alojamiento xmlns:q="http://empresaviajes.example.org/reserva/hoteles">
    <q:preferencia>ninguna</q:preferencia>
  </q:alojamiento>
</env:Body>
```

Ejemplo de comunicación SOAP

■ Intercambio de mensaje SOAP



Petición



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:gs="http://uniovi.com/soap/sdi/ws">
  <soapenv:Header/>
  <soapenv:Body>
    <gs:getMarksRequest>
      <gs:dni>75999999X</gs:dni>
    </gs:getMarksRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getMarksResponse xmlns:ns2="http://uniovi.com/soap/ws">
      <ns2:user>
        <ns2:dni>75999999X</ns2:dni>
        <ns2:name>Jose</ns2:name>
        <ns2:mark>
          <ns2:description>SDI</ns2:description>
          <ns2:score>10</ns2:score>
        </ns2:mark>
        <ns2:mark>
          <ns2:description>DLP</ns2:description>
          <ns2:score>8</ns2:score>
        </ns2:mark>
        <ns2:mark>
          <ns2:description>IP</ns2:description>
          <ns2:score>8</ns2:score>
        </ns2:mark>
      </ns2:user>
    </ns2:getMarksResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Respuesta



WSDL

(WEB SERVICES DESCRIPTION LANGUAGE)

Autodescripción de los servicios web - WSDL

- Un servicio web SOAP van acompañado de un descriptor WSDL
 - WSDL es el acrónimo de *Web Services Description Language*.
 - Es un *lenguaje basado en XML* utilizado para *describir la funcionalidad* que proporciona un servicio web.
 - Estandariza la *representación de los parámetros de entrada y salida* de un servicio web.
 - Permite que distintos clientes pueda entender como interactuar con un servicio web, automáticamente.
 - Independencia de la plataforma y lenguaje.

Descripción de servicios con WSDL

- Correspondencia sencilla con los sistemas de tipos de cualquier lenguaje de programación moderno.
- Tipo de datos:
 - Tipos Simples
 - Tipos Complejos (objetos)
 - Tipos Enumeración, Array

SOAP/XML
xsd:base64Binary
xsd:boolean
xsd:byte
xsd:dateTime
xsd:decimal
xsd:double
xsd:float
xsd:hexBinary
xsd:int
xsd:integer
xsd:long
xsd:QName
xsd:short
xsd:string

Estructura de un documento WSDL

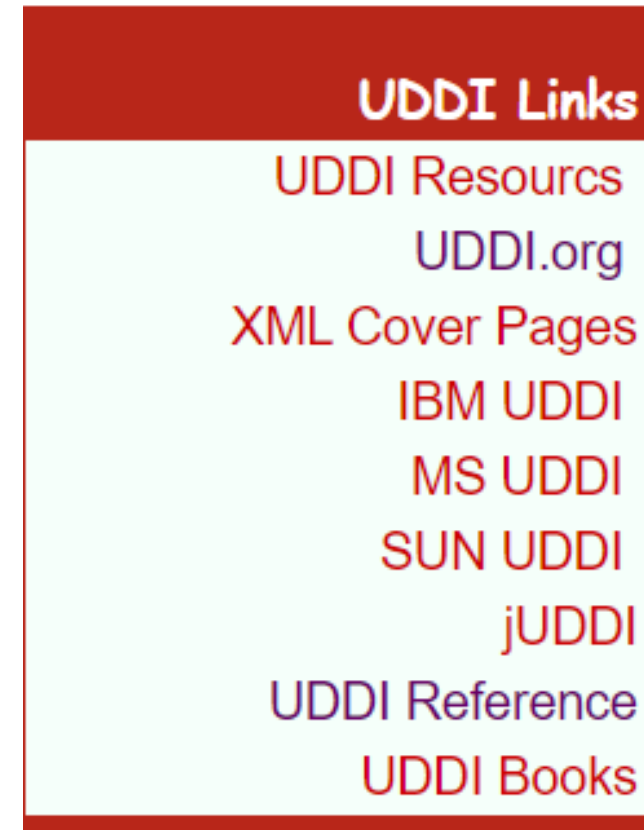
■ Ejemplo WSDL "hola mundo"

```
<message name="hello">
  <part name="param" type="xs:string"/>
</message>
<message name="helloResponde">
  <part name="valor" type="xs:string"/>
</message>
<portType name="HelloBean">
  <operation name="sayHello">
    <input message="hello"/>
    <output message="helloResponse"/>
  </operation>
</portType>
<binding type="GreeterPortBinding" name="helleBean">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="sayHello">
    <soap:operation soapAction="http://uoc.edu/obtTermino"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

UDDI
(UNIVERSAL DESCRIPTION, DISCOVERY AND
INTEGRATION)

UDDI (Universal Description, Discovery and Integration)

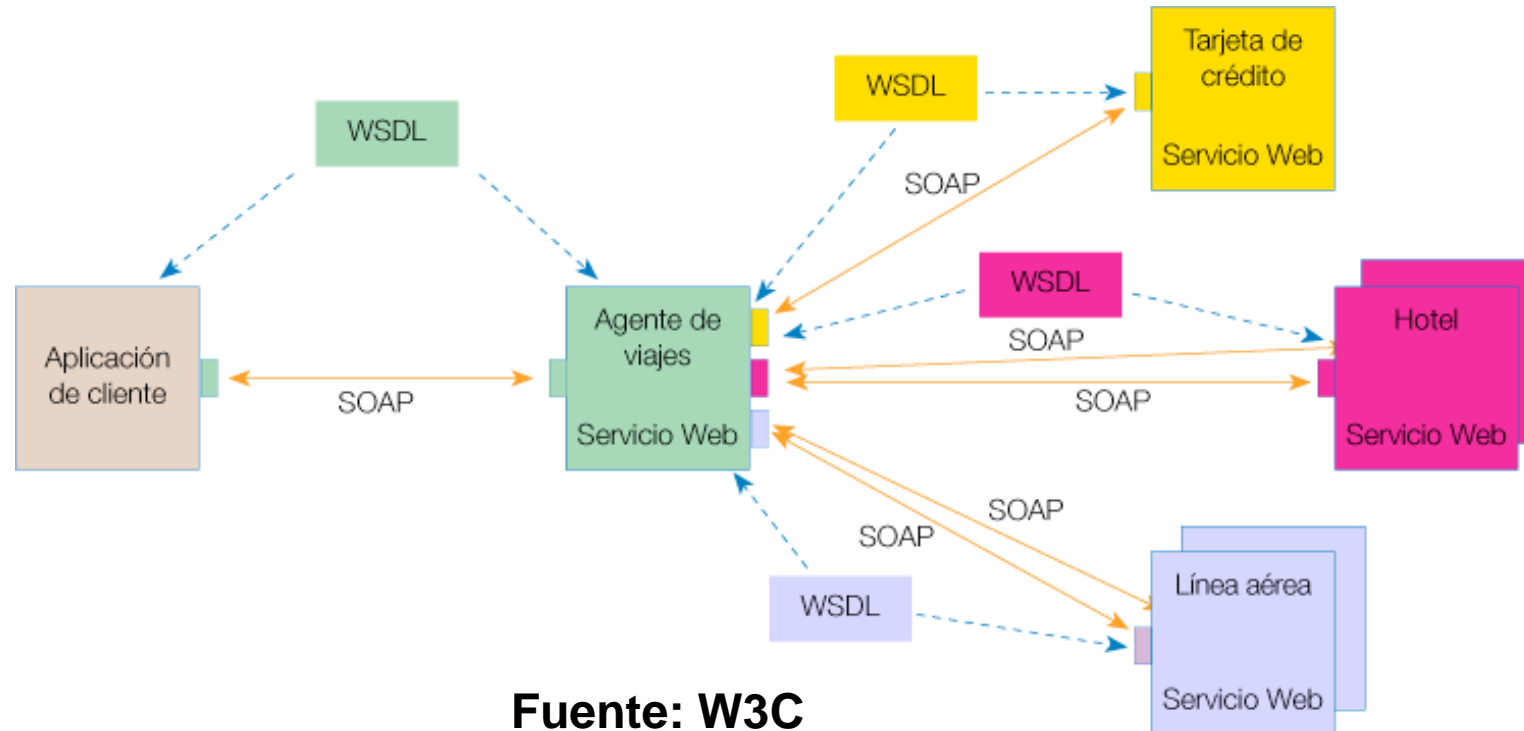
- UDDI (Universal Description, Discovery and Integration)
 - Definido para localizar servicios web.
 - Define la especificación para construir un **directorio distribuido de Servicios Web**, donde los datos se almacenan en XML.
 - Gestor de servicios que proporciona
 - Información sobre servicios.
 - Organizaciones que los proporcionan.
 - Categoría en la que se encuentran.
 - Instrucciones de uso (normalmente WSDL).



<http://soapclient.com/uddisearch.html>

Interoperabilidad entre múltiples servicios web

- SOAP está pensado para entornos distribuidos donde en una comunicación puede intervenir más de un nodo o servicio



Ventajas de un servicio web SOAP

- Basado en estándares
- Permite la interoperabilidad
 - Es independiente del lenguaje de programación, ya que XML es interoperable entre distintas tecnologías.
- Es versátil - Separación entre contenido y transporte
 - SOAP se enfoca en el formato de los datos (XML) y no en el protocolo de transporte.
 - Permite usar diferentes protocolos de transportes: HTTP, SMTP, JMS, etc.
- Es escalable
 - Si lo utilizamos junto HTTP resulta escalable y sencillo de utilizar en sistemas que contienen Firewalls o Proxies.
- Flexible y extensible
 - Permite a los desarrolladores personalizar y ampliar las funcionalidades del protocolo según sus necesidades.
 - Tiene varias extensiones comúnmente utilizadas para mejorar aspectos como la seguridad (WS-Security), enrutamiento (WS-Addressing), etc.
- Seguro
 - Ideal para la realización transacciones complejas y que requieren alta seguridad.

Desventajas de servicios web SOAP

- Rendimiento
 - Como se basa en una gramática XML puede ser considerablemente más lento que otros sistemas más ligeros CORBA, DCOM o RESTful.
 - Los binarios se codifican como texto, si se intenta transmitir un fichero grande puede penalizar el rendimiento (aunque está bastante optimizado).
- Complejidad
 - Desarrollo e interpretación puede ser compleja sino no se cuenta con las herramientas adecuadas.
- Dependencia del WSDL (Web Services Description Language).
- Menos flexible que REST
 - Puede no ser la mejor opción para aplicaciones web más simples.

SOAP vs REST

	SOAP	REST
Acrónimo	Simple Object Access Protocol.	Representational State Transfer.
Diseño	Protocolo estandarizado con reglas predefinidas y estrictas a seguir.	Estilo arquitectónico con pautas y recomendaciones sueltas.
Enfoque	Basado en funciones (datos disponibles como servicios, por ejemplo: "getUser").	Basado en datos (datos disponibles como recursos, por ejemplo, "usuario").
Estado	Sin estado de forma predeterminada, pero es posible hacer que una API de SOAP tenga estado.	Sin estado (sin sesiones del lado del servidor).
Almacenamiento en caché	Las llamadas API no se pueden almacenar en caché.	Las llamadas API se pueden almacenar en caché.
Seguridad	Ofrece funciones de seguridad avanzada. WS-Security con soporte SSL. Cumplimiento ACID incorporado.	Soporta HTTPS y SSL.
Rendimiento	Requiere más ancho de banda y potencia de computación.	Más ligero. Requiere menos ancho de banda y recursos.
Formato de mensaje	Solo XML	HTML, XML, JSON, y otros.
Protocolo(s) de transferencia	HTTP, SMTP, UDP y otros.	Solo HTTP
Tipado	Fuertemente tipado (WSDL).	Débilmente tipado.
Ventajas	Alta seguridad, estandarizada, extensible.	Escalabilidad, mejor rendimiento, facilidad de navegación, flexibilidad.
Desventajas	Menor rendimiento, más complejidad, menos flexibilidad	Menos seguridad.

Fuente: Basado en <https://raygun.com/blog/soap-vs-rest-vs-json/>

IMPLEMENTACIÓN ARQUITECTURA SOAP

Arquitectura SOAP

- Vamos a ver como desarrollar un servidor y un cliente de **servicios web SOAP**, usando **Spring boot**.
- En primer lugar, desarrollaremos un **servicio web que va a exponer datos** de las notas de las asignaturas de los alumnos.
- En segundo lugar, construiremos un **cliente que obtenga datos del servicio web** anterior a partir del WSDL generado.

Servicio web SOAP con Spring

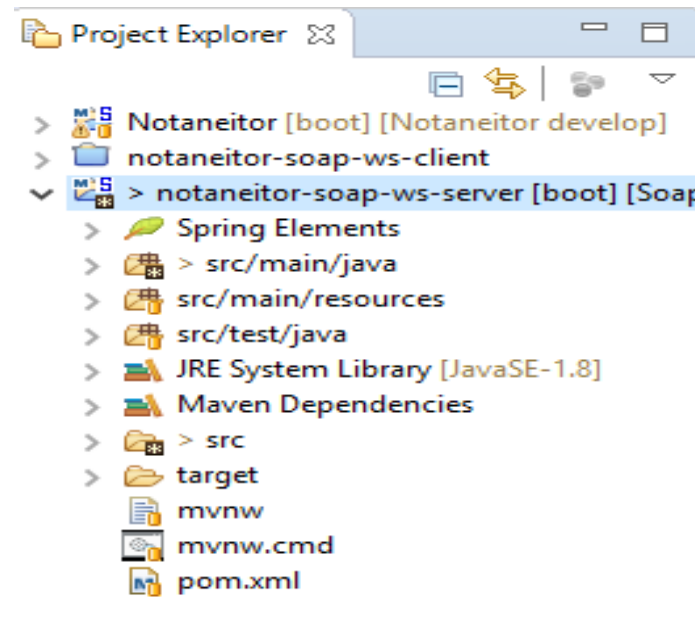
- El servicio web Spring proporciona un acoplamiento flexible entre el contrato y la implementación.
- Proporciona un potente mapeo entre la solicitud XML entrante y cualquier objeto.
- Utiliza potentes APIs para Serializar y deserializar mensajes XML entrantes.
- La seguridad del servicio spring web permite firmar mensajes SOAP, encriptarlos y desencriptarlos.

Servicio web SOAP

- Pasos para crear un servicio web SOAP con spring boot
 - Crear un proyecto spring-boot
 - Añadir las dependencias Spring-WS
 - Crear un XML schema para definir el dominio
 - Generar clases de dominio basadas en el esquema XML
 - Crear un repositorio de datos
 - Crear los endpoints del servicio web
 - Configurar los beans para gestionar los servicios
 - Configurar URL base y puertos
 - Ejecutar la aplicación para obtener el WSDL
 - Probando el WSDL

Servicio web SOAP

- Paso 1. Crear un proyecto spring-boot



Servicio web SOAP

- Paso 2. Añadir Dependencias *spring-boot-starter-web-services y wsdl4j*
 - Permitirán generar la información *formato WSDL*.
 - Contendrá los mensajes, la ubicación, los enlaces y las operaciones disponibles.
 - Podemos utilizar también la dependencia *spring-ws-core*.

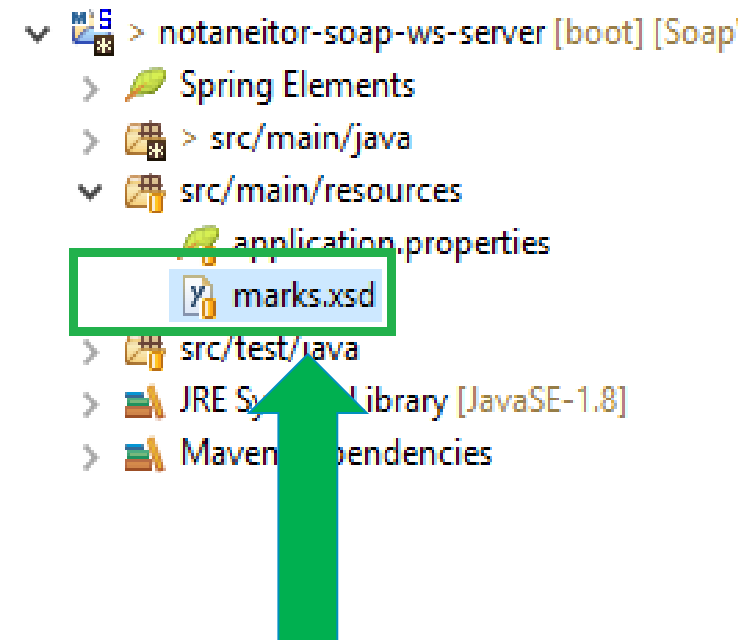
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web-services</artifactId>
</dependency>
<dependency>
  <groupId>wsdl4j</groupId>
  <artifactId>wsdl4j</artifactId>
</dependency>
```

Servicio web SOAP

- Paso 3. Crear un XML schema para definir el dominio
 - El dominio del servicio web se define en un archivo de XML schema (XSD) que describe en formato XML:
 - Los *elementos, atributos y operaciones* que ofrecerá el servicio web SOAP.
 - Spring *exportará automáticamente como WSDL* y que será lo que se consuma por un cliente.
 - Debe crearse dentro de la carpeta del proyecto spring *src/main/resources*

Servicio web SOAP

- mark.xsd definirá:
 - **getMarksRequest**: Método que permitirá hacer una petición al servicio web y que enviará como parámetro el dni del alumno.
 - **getMarksResponse**: Método que devolverá un elemento de tipo user.
 - **User**: Elemento que representa un objeto complejo que contiene el dni, nombre y notas (mark) de un usuario (alumno).
 - **Mark**: Elemento que representa un objeto complejo que contiene el descripción y puntuación(score) de una nota (podemos llamarle asignatura).




Servicio web SOAP

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://uniovi.com/soap/ws"
    targetNamespace="http://uniovi.com/soap/ws"
    elementFormDefault="qualified">
  <xs:element name="getMarksRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="dni" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="getMarksResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="user" type="tns:user"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```



Servicio web SOAP



```
<xs:complexType name="user">
  <xs:sequence>
    <xs:element name="dni" type="xs:string"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="mark" type="tns:mark"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="mark">
  <xs:sequence>
    <xs:element name="description" type="xs:string"/>
    <xs:element name="score" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Servicio web SOAP

- Paso 4. ***Generar clases*** del dominio basadas en un esquema XML.
 - El plugin de maven ***jaxb2-maven-plugin*** permite generar las clases Java automáticamente.
 - Este plugin usa la herramienta ***XJC*** como motor de generación de código.
 - XJC ***compila un archivo XML schema*** y ***genera las clases Java*** totalmente anotadas a partir de este fichero xsd.

Servicio web SOAP

■ Paso 4.1. Añadir plugin a las dependencias (POM.XML)

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>1.6</version>
  <executions>
    <execution>
      <id>xjc</id>
      <goals><goal>xjc</goal></goals>
    </execution>
  </executions>
  . . .
```

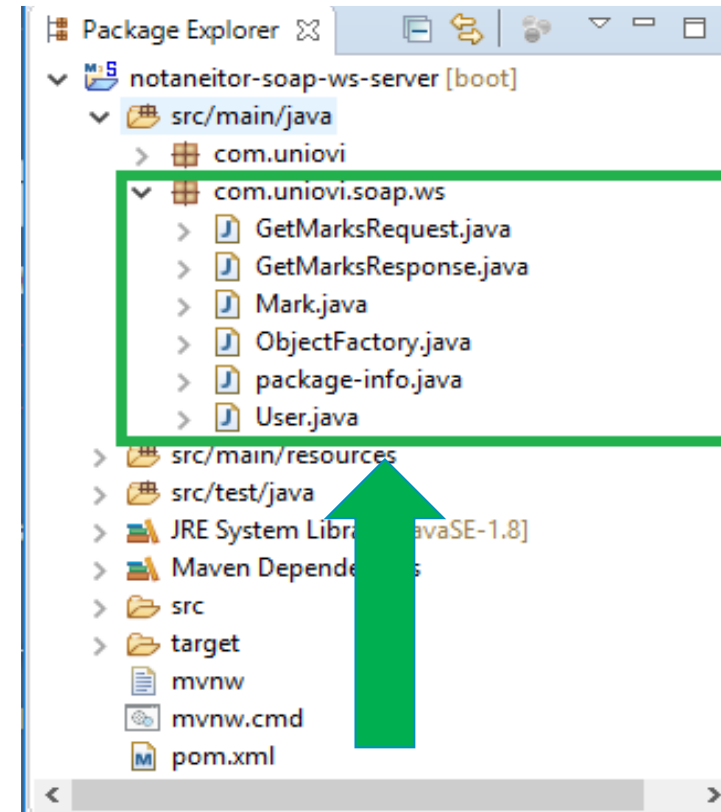
Servicio web SOAP

■ Paso 4.2. Configuración plugin

```
. . .  
<configuration>  
<schemaDirectory>${project.basedir}/src/main/resources/</schemaDirectory>  
[redacted]  
<outputDirectory>${project.basedir}/src/main/java</outputDirectory>  
<clearOutputDir>false</clearOutputDir>  
</configuration>  
. . .
```

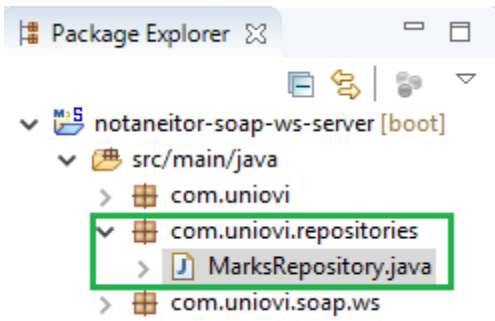
Servicio web SOAP

- Paso 4.3. Actualizar proyecto para generar clases JAVA
 - *Maven-> Update Project*
 - *Se genera una clase por cada elemento del fichero XML Schema*



Servicio web SOAP

- Paso 5. Crear un repositorio de datos
 - Proporcionará los datos al servicio web



```
package com.uniovi.repositories;

...

@Component
public class MarksRepository {

    private static final Map<String, User> marks =
new HashMap<>();

    @PostConstruct
    public void initData() {...}

    public User findAllByUser(String dni) {
        return marks.get(dni);
    }

}
```

Servicio web SOAP

- Paso 6. Crear un Endpoint
 - Los ENDPOINTS definen los puntos de entradas a los que los clientes podrán hacer peticiones al servicio web disponible.
 - En sprint boot, para definir un EndPoint hay que crear una clase y anotarla con la etiqueta *@Endpoint*
 - En esta clase hay que *definir y anotar los métodos controladores* que se ejecutarán cuando un cliente haga una petición al servicio web.

Servicio web SOAP

- Paso 6. Crear un Endpoint (Anotaciones)
 - **@Endpoint:** registra la clase con Spring WS. para que pueda procesar los mensajes SOAP entrantes.
 - **@PayloadRoot:** Elige el método que va a manejar la solicitud entrante
 - Basado en el espacio de nombres del mensaje (URL base de SW)
 - El parametro LocalPart definido en el servicio (método de la petición).
 - **@RequestPayload:** indica que el mensaje entrante será mapeado al parámetro de solicitud del método correspondiente.
 - **@ResponsePayload:** hace que Spring WS asigne el valor devuelto a la respuesta Payload (mensaje)

Servicio web SOAP

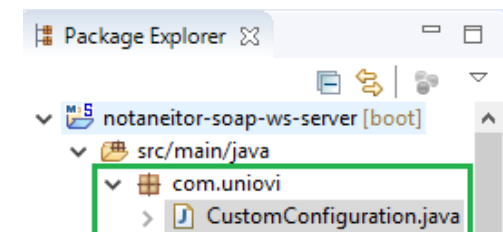
```
@Endpoint
public class MarkEndpoint {
    private static final String NAMESPACE_URI = "http://uniovi.com/soap/ws";
    private MarksRepository markRepository;
    @Autowired
    public MarkEndpoint(MarksRepository markRepository) {
        this.markRepository = markRepository;
    }
    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getMarksRequest")
    @ResponsePayload
    public GetMarksResponse getMarks(@RequestPayload GetMarksRequest request) {
        GetMarksResponse response = new GetMarksResponse();
        response.setUser(markRepository.findAllByUser(request.getDni()));
        return response;
    }
}
```



Servicio web SOAP

- Paso 7. Creamos una nueva clase de configuración del servicio web
 - Tiene que estar anotada con **@EnableWs** y heredar de **WsConfigurerAdapter**
 - **@EnableWs**: proporciona la configuración del servicio web de Spring.
 - **WsConfigurerAdapter**: es una clase adaptador que contiene los métodos necesarios para configurar los beans relacionados con los servicios web de Spring.

```
@EnableWs
@Configuration
public class CustomConfiguration extends WsConfigurerAdapter
{
```



Servicio web SOAP

- Paso 7.1. Definir los Beans:
 - **MessageDispatcherServlet:** Tipo de servlet utilizado por spring WS para manejar mensajes SOAP
 - **XsdSchema:** Abstracción para el esquema XSD definido.
 - **DefaultWsd11Definition:** Crea SOAP para el esquema XSD dado.
- Estos bean son los que determinan la URL bajo la cual el servicio web y el archivo WSDL generado estarán disponibles
 - URL en la que fichero WSDL estará disponible:
 - http: //<host> :<Puerto> /webservice/marks.wsdl.
 - Ejemplo:
 - http://localhost:8090/**webservice/marks.wsdl**.

Servicio web SOAP



- Paso 7.2. Bean *MessageDispatcherServlet*
 - Hay que inyectar y establecer *ApplicationContext* a *MessageDispatcherServlet* para que Spring WS *detecte el bean automáticamente*

```
@EnableWs
@Configuration
public class CustomConfiguration extends WsConfigurerAdapter {
    @Bean
    public ServletRegistrationBean messageDispatcherServlet(ApplicationContext
applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean(servlet, "/webservice/*");
    }
}
```

Servicio web SOAP

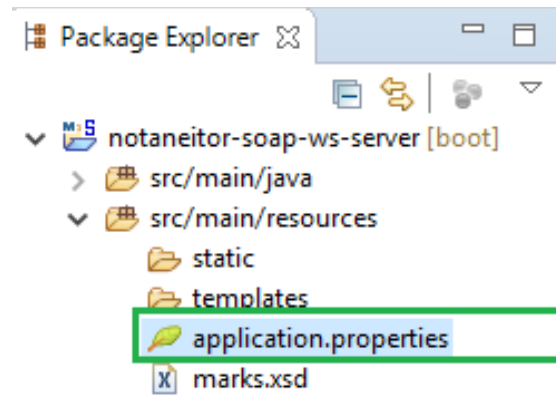
■ Paso 7.3. Bean DefaultWsd11Definition

```
@EnableWs
@Configuration
public class CustomConfiguration extends WsConfigurerAdapter {
    . . .
    @Bean(name = "marks")
    public DefaultWsd11Definition defaultWsd11Definition(XsdSchema marksSchema) {
        DefaultWsd11Definition wsdl11Definition = new DefaultWsd11Definition();
        wsdl11Definition.setPortTypeName("MarksPort");
        wsdl11Definition.setLocationUri("/webservice/marks");
        wsdl11Definition.setTargetNamespace("http://uniovi.com/soap/ws");
        wsdl11Definition.setSchema(marksSchema);
        return wsdl11Definition;
    }
    @Bean
    public XsdSchema marksSchema() {
        return new SimpleXsdSchema(new ClassPathResource("marks.xsd"));
    }
}
```



Servicio web SOAP

- Paso 8. Configurar el Puerto
 - Modificamos el fichero *application.properties* de la aplicación

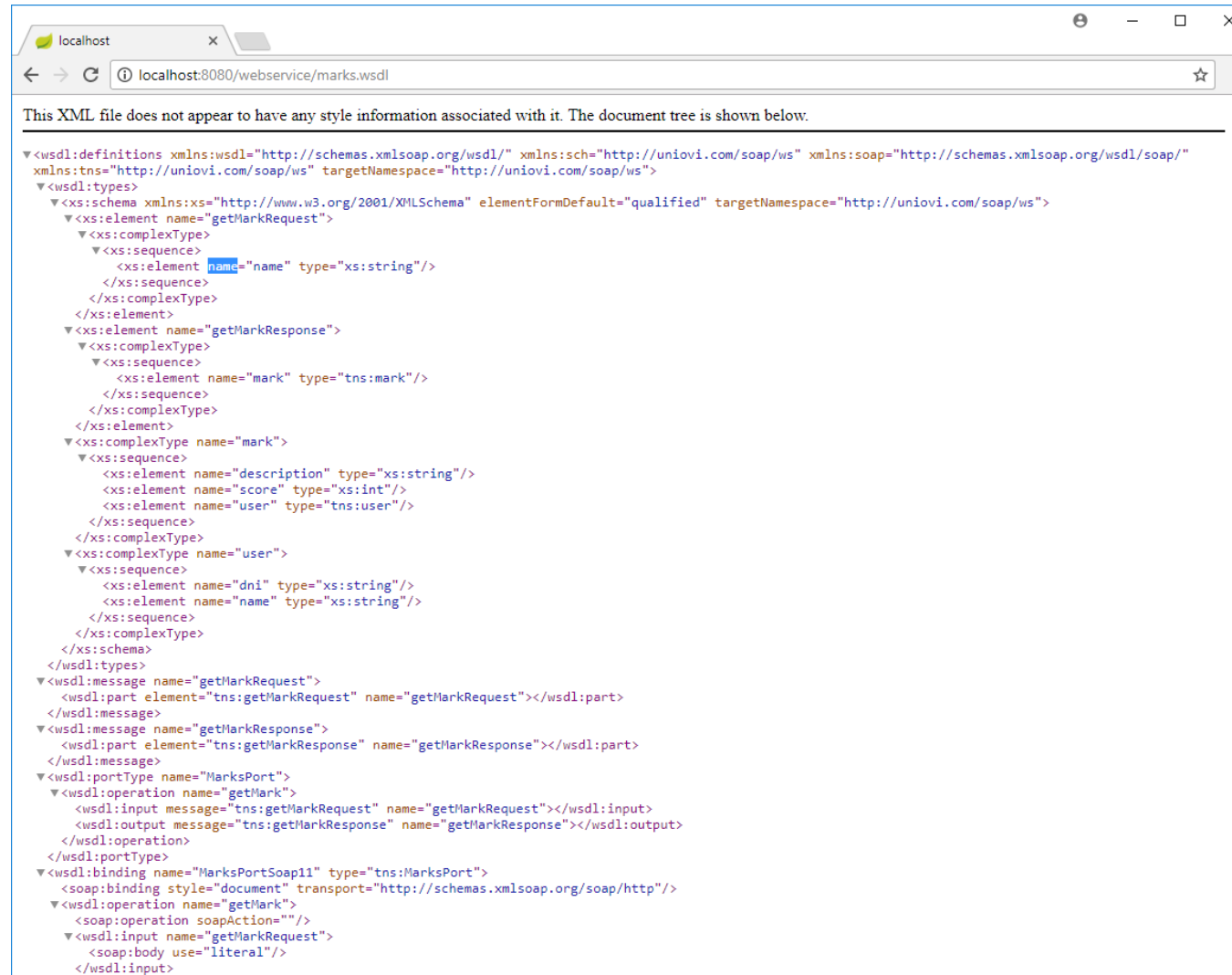


server.port=8090

Servicio web SOAP

- Paso 9. Ejecutar la aplicación y obtener el WSDL
 - Ejecutamos la aplicación
 - Accedemos la URL del servicio de uun navegador
 - <http://localhost:8090/webservice/marks.wsdl>

Servicio web SOAP



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch="http://uniovi.com/soap/ws" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://uniovi.com/soap/ws" targetNamespace="http://uniovi.com/soap/ws">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://uniovi.com/soap/ws">
      <xs:element name="getMarkRequest">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getMarkResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="mark" type="tns:mark"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="mark">
        <xs:sequence>
          <xs:element name="description" type="xs:string"/>
          <xs:element name="score" type="xs:int"/>
          <xs:element name="user" type="tns:user"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="user">
        <xs:sequence>
          <xs:element name="dni" type="xs:string"/>
          <xs:element name="name" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getMarkRequest">
    <wsdl:part element="tns:getMarkRequest" name="getMarkRequest"/>
  </wsdl:message>
  <wsdl:message name="getMarkResponse">
    <wsdl:part element="tns:getMarkResponse" name="getMarkResponse"/>
  </wsdl:message>
  <wsdl:portType name="MarksPort">
    <wsdl:operation name="getMark">
      <wsdl:input message="tns:getMarkRequest" name="getMarkRequest"/>
      <wsdl:output message="tns:getMarkResponse" name="getMarkResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="MarksPortSoap11" type="tns:MarksPort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getMark">
      <soap:operation soapAction="">
        <wsdl:input name="getMarkRequest">
          <soap:body use="literal"/>
        </wsdl:input>
      </soap:operation>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

Servicio web SOAP

- Ejemplo operaciones disponibles



```
▼<wsdl:portType name="MarksPort">
  ▼<wsdl:operation name="getMark">
    <wsdl:input message="tns:getMarkRequest" name="getMarkRequest"></wsdl:input>
    <wsdl:output message="tns:getMarkResponse" name="getMarkResponse"></wsdl:output>
  </wsdl:operation>
</wsdl:portType>
▼<wsdl:binding name="MarksPortSoap11" type="tns:MarksPort">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  ▼<wsdl:operation name="getMark">
    <soap:operation soapAction=""/>
    ▼<wsdl:input name="getMarkRequest">
      <soap:body use="literal"/>
    </wsdl:input>
```

Servicio web SOAP

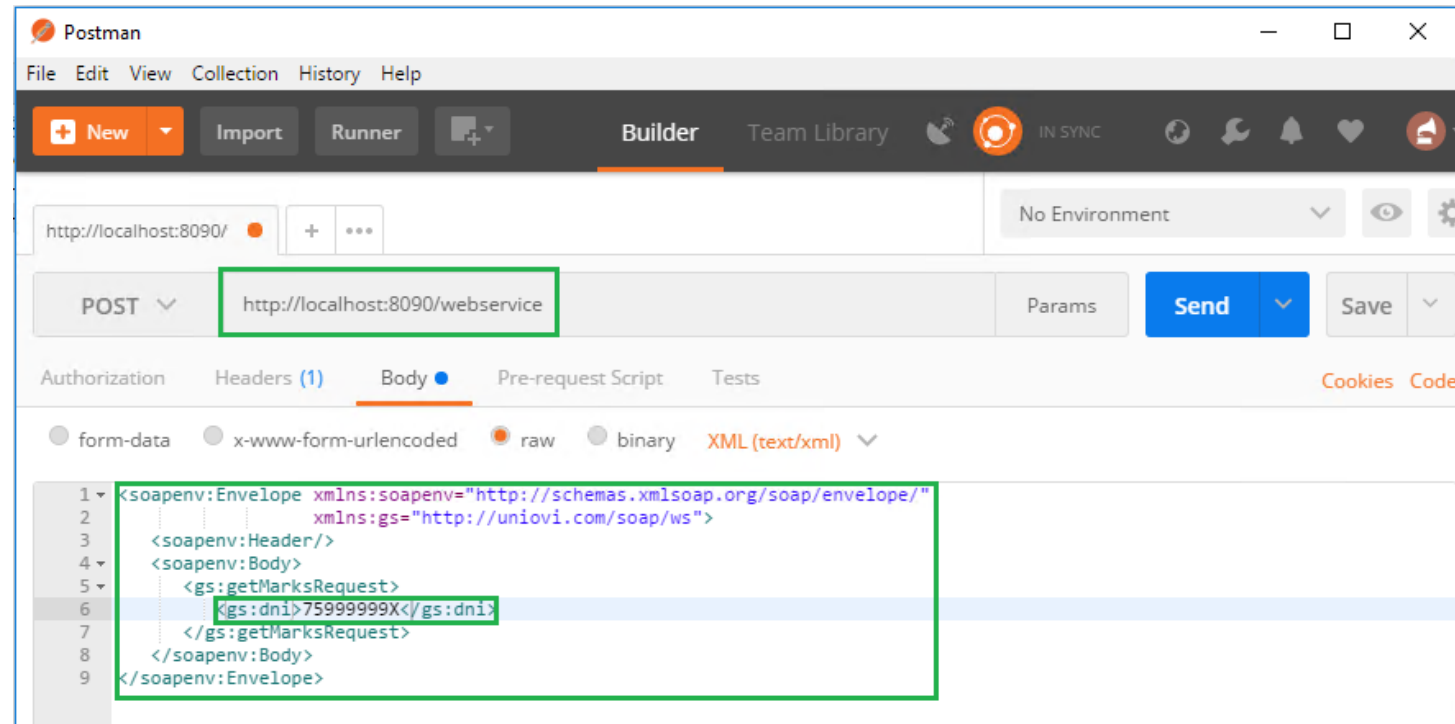
- Paso 10. Probando el servicio we WSDL
 - Usar POSTMAN para hacer la petición POST
 - Configurar el **HEADER** como "***content-type: text/xml***"
 - Pasar cuerpo del mensaje pasarle un sobre(***soapenv:Envelope***)



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
                  xmlns:gs="http://uniovi.com/soap/ws">  
  <soapenv:Header/>  
  <soapenv:Body>  
    <gs:getMarksRequest>  
      <gs:dni>75999999X</gs:dni>  
    </gs:getMarksRequest>  
  </soapenv:Body>  
</soapenv:Envelope>
```

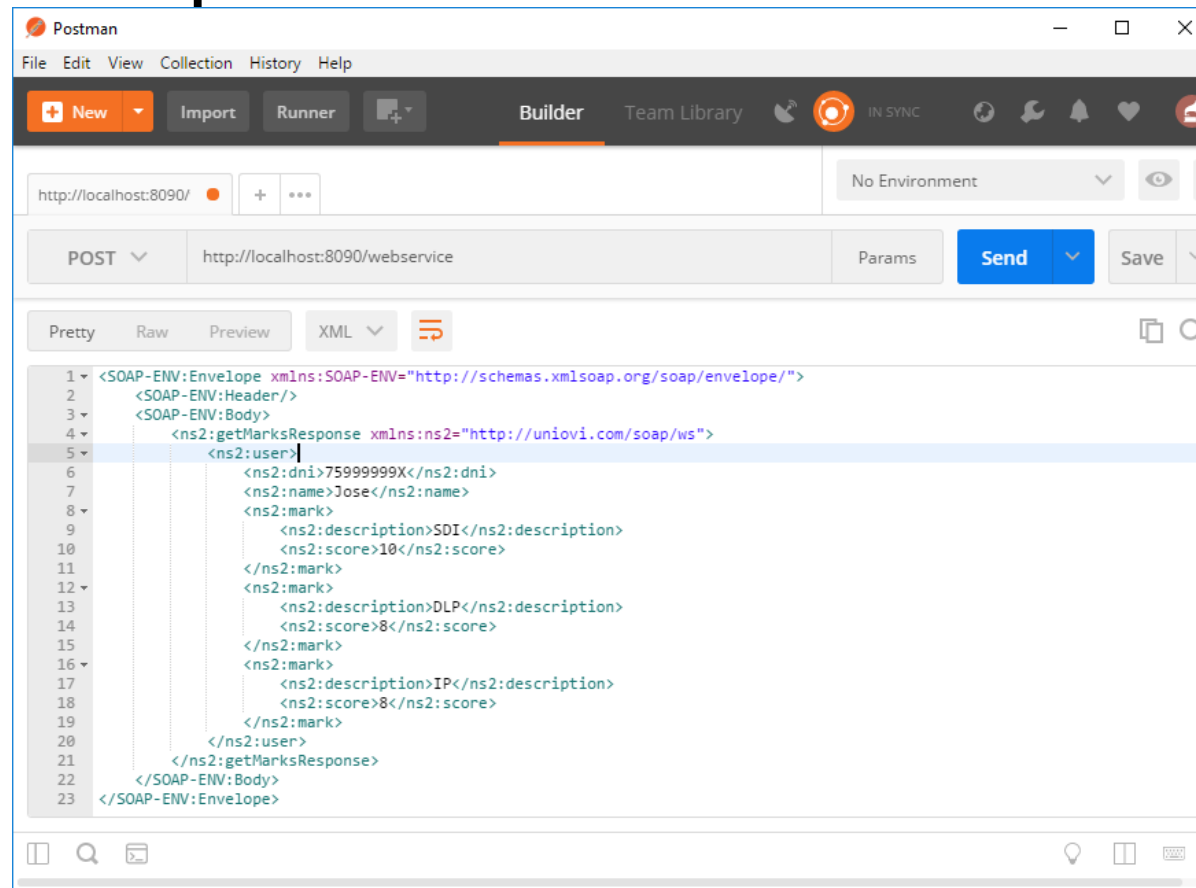

Servicio web SOAP

■ Paso 10.1. Petición



Servicio web SOAP

■ Paso 10.1. Respuesta



The screenshot shows the Postman application window. The URL bar is set to `http://localhost:8090/`. The request method is `POST` and the endpoint is `http://localhost:8090/webservice`. The response is displayed in the 'Pretty' view, showing an XML document. The XML structure is as follows:

```
1 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
2   <SOAP-ENV:Header/>
3   <SOAP-ENV:Body>
4     <ns2:getMarksResponse xmlns:ns2="http://uniovi.com/soap/ws">
5       <ns2:user>
6         <ns2:dni>75999999X</ns2:dni>
7         <ns2:name>Jose</ns2:name>
8         <ns2:mark>
9           <ns2:description>SDI</ns2:description>
10          <ns2:score>10</ns2:score>
11        </ns2:mark>
12        <ns2:mark>
13          <ns2:description>DLP</ns2:description>
14          <ns2:score>8</ns2:score>
15        </ns2:mark>
16        <ns2:mark>
17          <ns2:description>IP</ns2:description>
18          <ns2:score>8</ns2:score>
19        </ns2:mark>
20      </ns2:user>
21    </ns2:getMarksResponse>
22  </SOAP-ENV:Body>
23 </SOAP-ENV:Envelope>
```

IMPLEMENTACIÓN CLIENTE SERVICIO WEB SOAP

Cliente servicio web SOAP

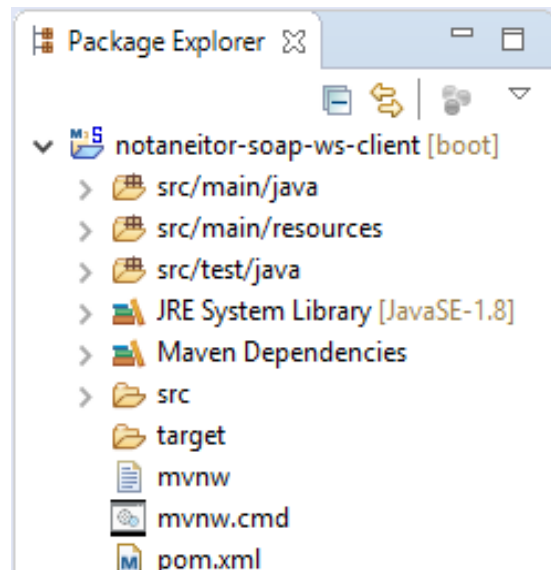
- Construiremos una aplicación web cliente que obtenga datos publicados de un servicio web SOAP
- En nuestro caso, consumiremos los datos del servicio web de notas desarrollado anteriormente.
 - <http://localhost:8090/webservice/marks.wsdl>

Servicio web SOAP

- Pasos para crear un servicio web SOAP cliente con spring boot
 - Crear un proyecto spring-boot
 - Añadir las dependencias Spring-WS
 - Generar los objetos del dominio (clases JAVA) basado en el WSDL
 - Definir una clase que herede de la clase WebServiceGatewaySupport y definir sus operaciones.
 - En el fichero applications.properties definir puerto, endpoint y métodos del servicio web a utilizar
 - Definir clase de configuración para serializar y deserializar peticiones XML
 - Crear controladores y vistas a utilizar en la aplicación
 - Probar aplicación.

Cliente servicio Web SOAP

- Paso 1. Crear un proyecto spring-boot



Cliente servicio web SOAP

- Paso 2. Añadir a dependencia al fichero pom.xml:
 - spring-ws-core o
 - spring-boot-starter-ws

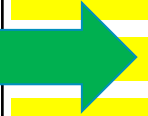
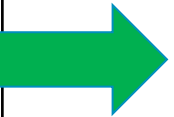
```
<dependency>  
  <groupId>org.springframework.ws</groupId>  
  <artifactId>spring-ws-core</artifactId>  
</dependency>
```

Cliente servicio web SOAP

- Paso 3. ***Generar clases*** de dominio basadas en WSDL.
 - El plugin de maven ***maven-jaxb2-plugin.***
 - Permite generar las clases Java automáticamente a partir de un fichero WSDL

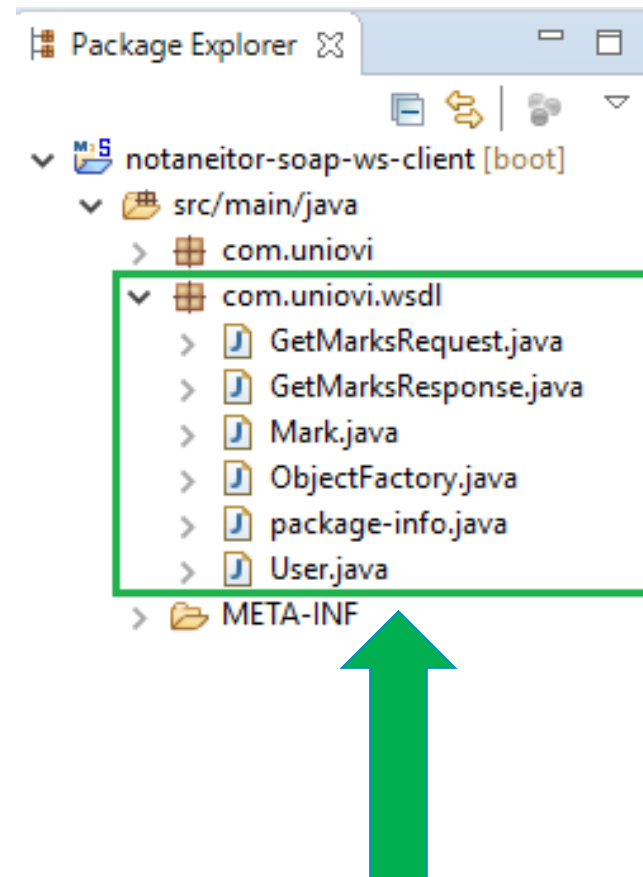
Cliente servicio web SOAP

```
<plugin>
  <groupId>org.jvnet.jaxb2.maven2</groupId>
  <artifactId>maven-jaxb2-plugin</artifactId>
  <version>0.13.1</version>
  <executions>
    <execution> <goals> <goal>generate</goal></goals></execution>
  </executions>
  <configuration>
    <schemaLanguage>WSDL</schemaLanguage>
    <generatePackage>com.uniovi.wsdl</generatePackage>
    <generateDirectory>${project.basedir}/src/main/java</generateDirectory>
    <clearOutputDir>false</clearOutputDir>
    <schemas>
      <schema>
        <url>http://localhost:8090/webservice/marks.wsdl</url>
      </schema>
    </schemas>
  </configuration>
</plugin>
```



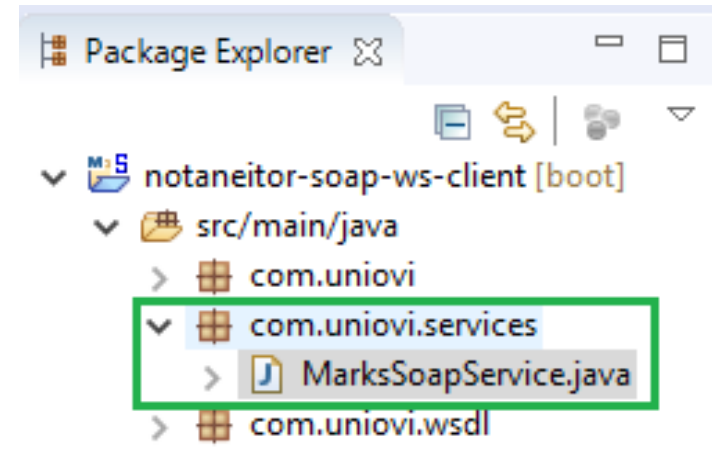
Cliente servicio web SOAP

- Paso 4.3. Actualizar proyecto para generar clases JAVA
 - *Maven-> Update Project*
 - *Se generan las clases JAVA correspondiente*
 - *Nota: El servicio web SOAP del que obtendremos el WSDL debe estar disponible.*



Cliente servicio web SOAP

- Paso 5. Crear servicio cliente
 - Crear una clase de cliente que herede de **WebServiceGatewaySupport** y definir sus operaciones.
 - WebServiceGatewaySupport es una superclase que facilita que una sub-clase pueda acceder a los métodos de un servicio web.



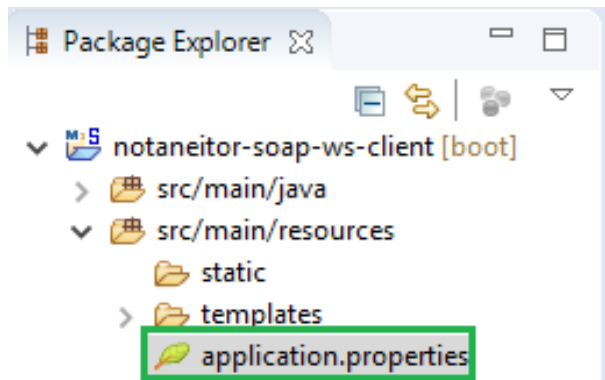
Cliente servicio web SOAP



```
public class MarksSoapService extends WebServiceGatewaySupport {  
    @Value("${service.endpoint}")  
    private String serviceEndpoint;  
    @Value("${service.soap.action}")  
    private String serviceSoapAction;  
    public GetMarksResponse getMarks(String dni) {  
        GetMarksRequest request = new GetMarksRequest();  
        request.setDni(dni);  
        GetMarksResponse response = (GetMarksResponse)  
        getWebServiceTemplate().marshalSendAndReceive(serviceEndpoint,  
            request, new SoapActionCallback(serviceSoapAction));  
        return response;  
    }  
}
```

Cliente servicio web SOAP

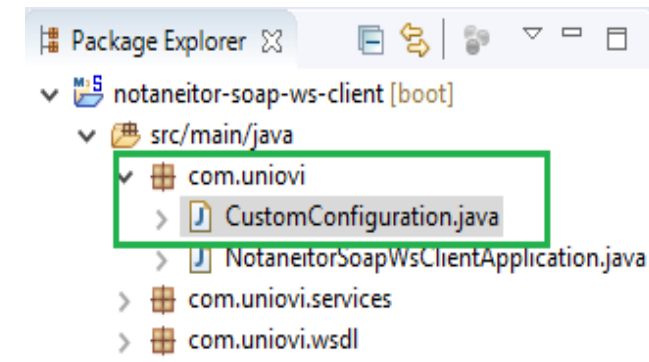
- Paso 6. Modificar el fichero applications.properties
 - Configurar puerto, endpoint y métodos que se utilizan para invocar el servicio web SOAP disponible en estas urls.



```
server.port=8091  
service.endpoint=http://localhost:8090/webservice  
service.soap.action=http://localhost:8090/webserv  
ice/GetMark
```

Ciente servicio web SOAP

- Paso 7. Serializar y deserializar las peticiones
 - Los servicio web SOAP responden en formato XML
 - Spring utiliza la implementación ***Jaxb2Marshaller*** para serializar y deserializar las solicitudes XML.
 - Definimos nueva clase ***CustomConfiguration*** que incluya los beans necesarios para serializar las peticiones.



Cliente servicio web SOAP

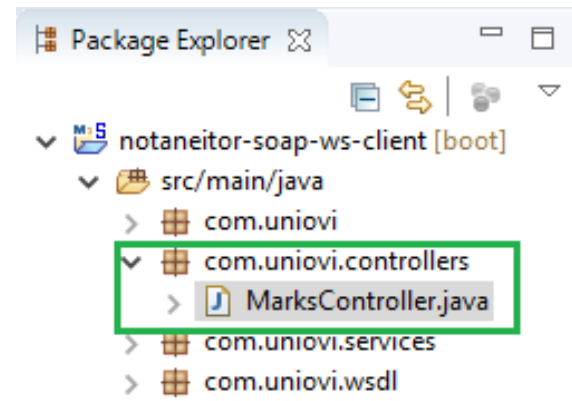
- Necesitamos 2 beans para serializar y deserializar las peticiones, en nuestro caso:
 - El serializador (**marshaller**) que apunta a la colección de **objetos de dominio generados** y los utilizará para **serializar y deserializar** entre XML y POJOs.
 - El servicio (**marksService**): este bean se crea y configura con la URI del servicio a utilizar y con el serializador (marshaller).

Cliente servicio web SOAP

```
@Configuration
public class CustomConfiguration {
    @Value("${service.endpoint}")
    private String serviceEndpoint;
    @Bean
    public Jaxb2Marshaller marshaller() {
        Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
        marshaller.setContextPath("com.uniovi.wsdl");
        return marshaller;
    }
    @Bean
    public MarksSoapService marksService(Jaxb2Marshaller marshaller) {
        MarksSoapService client = new MarksSoapService();
        client.setDefaultUri(serviceEndpoint);
        client.setMarshaller(marshaller);
        client.setUnmarshaller(marshaller);
        return client;
    }
}
```


Cliente servicio web SOAP

- Paso 8. Crear controlador
 - Hay que crear un controlador que se encargue de recibir las peticiones del cliente, llamar al servicio web ***MarksSoapService*** creado previamente y generar una respuesta.



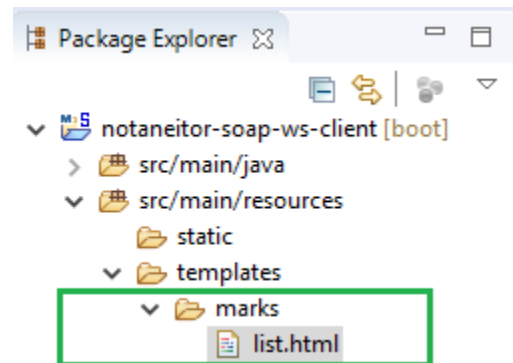
Cliente servicio web SOAP

```
@Controller
public class MarksController {
    @Autowired
    private MarksSoapService marksSoapService;

    @RequestMapping("/marks/list")
    public String getMarks(Model model, @RequestParam String dni) {
        List<Mark> marks = new ArrayList<Mark>();
        User user = marksSoapService.getMarks(dni).getUser();
        if (user != null) {
            marks = user.getMark();
        }
        model.addAttribute("dni", dni);
        model.addAttribute("markList", marks);
        return "marks/list";
    }
}
```

Cliente servicio web SOAP

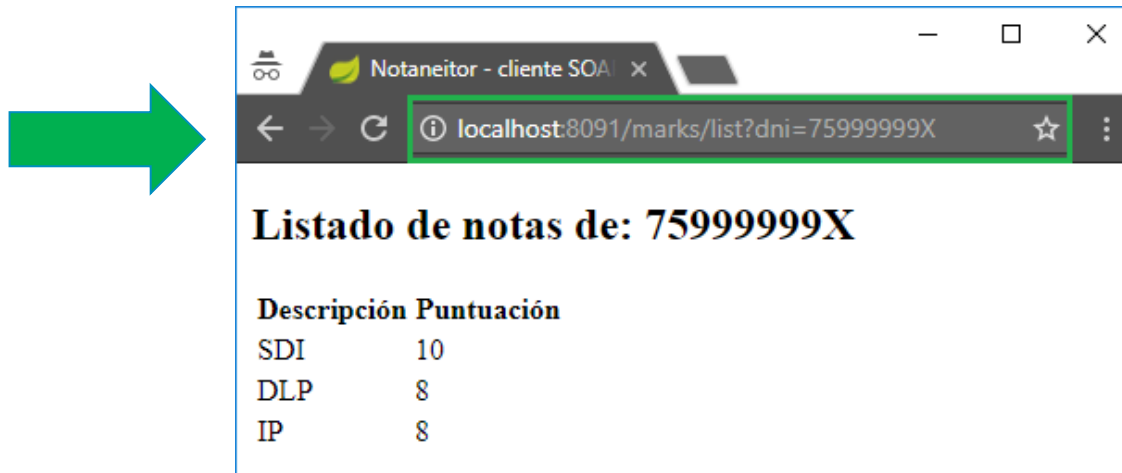
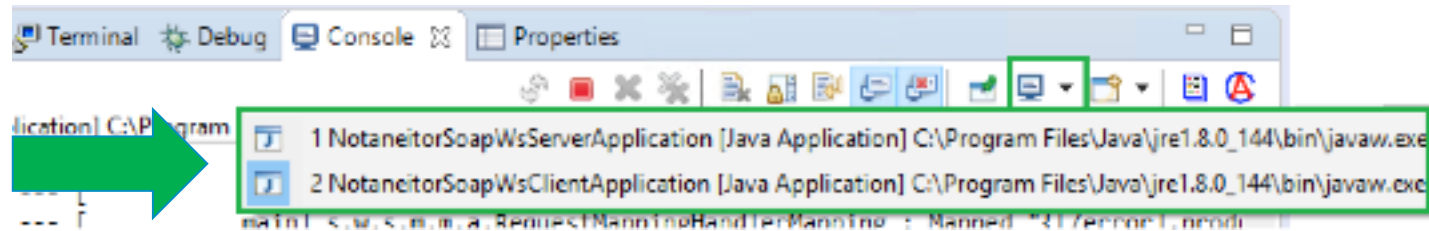
- Paso 9. Crear vista
 - Crear la vista para mostrar los datos que nos devuelve el controlador en formato HTML



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Notaneitor - cliente SOAP</title>
<meta charset="utf-8" />
</head>
<body>
  <div class="container">
    <h2>
      Listado de notas de: <span th:text="${dni}">99999999K</span>
    </h2>
    <table>
      <thead>
        <tr>
          <th>Descripción</th>
          <th>Puntuación</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="mark : ${markList}">
          <td th:text="${mark.description}">asignatura</td>
          <td th:text="${mark.score}">10</td>
        </tr>
      </tbody>
    </table>
    <div th:if="${#lists.isEmpty(markList)}">Lista de notas vacía</div>
  </div>
</body>
</html>
```

Cliente servicio web SOAP

■ Paso 9. Probando el servicio web cliente





Escuela de Ingeniería Informática

Escuela de Ingeniería Informática
School of Computer Science Engineering

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Sistemas Distribuidos e Internet

Tema 9 Servicios Web SOAP

SOAP
Contract First Web Services

RESTful API
GET PUT POST DELETE

Dr. Edward Rolando Núñez Valdez

nunezedward@uniovi.es