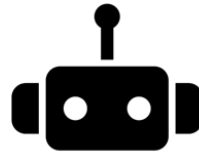




PUCP

SESIÓN DE LABORATORIO 2

Soap y Rest



HORARIO 10M1

Empezaremos a las 7:10 p.m

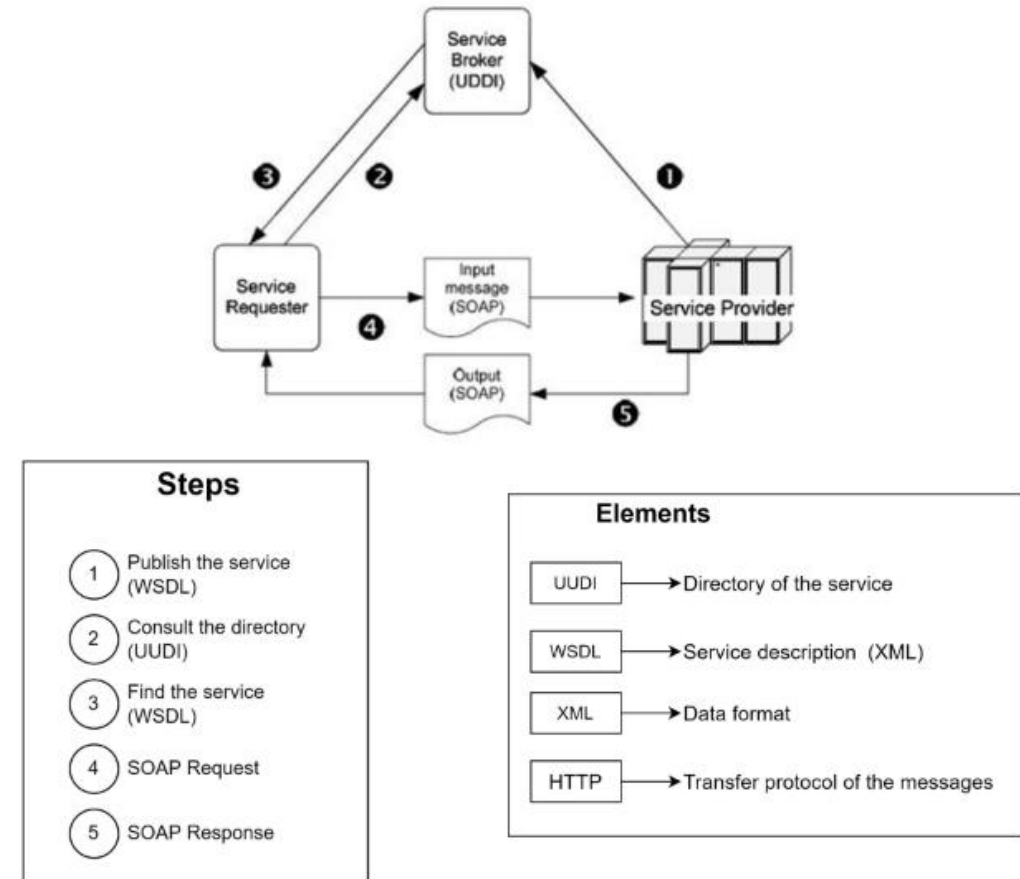
Gracias!



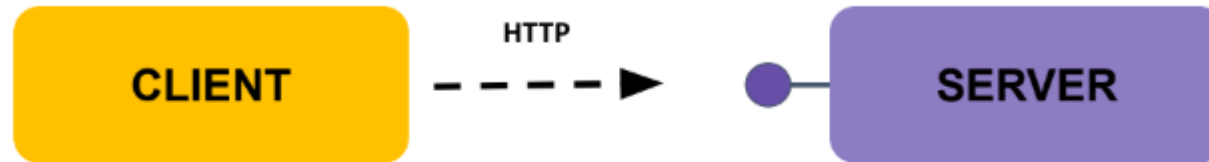
Que es SOAP?

Es un protocolo estándar que incluye reglas para intercambio de mensajes usando HTTP.

Los mensajes SOAP necesitan ser formateados como documentos XML



Que es REST?



Representational State Transfer

Operaciones CRUD



CRUD Operations

REST VS SOAP



REST

- Social Media
- Web Chat
- Mobile

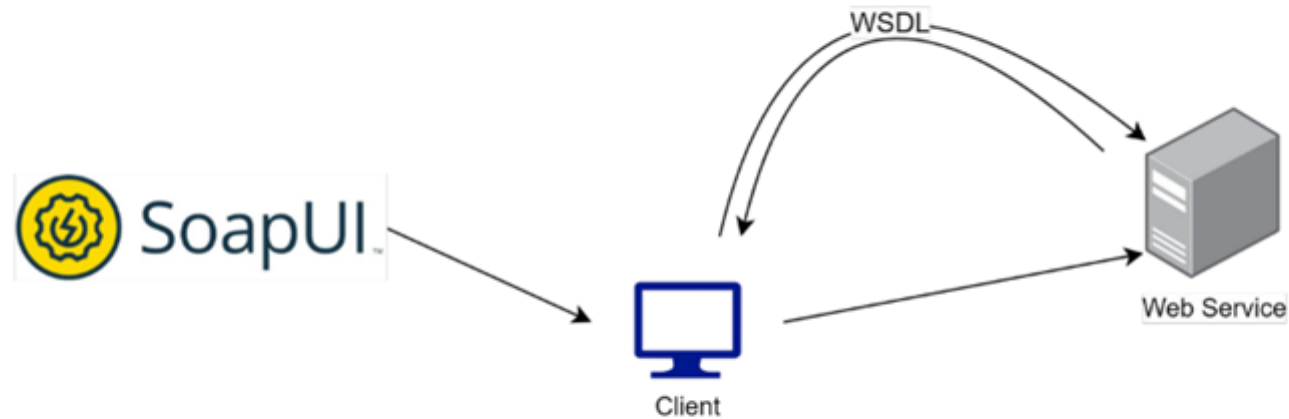


SOAP

- Financial
- Telecommunication
- Payment Gateways

Desarrollo de nuestra aplicación SOAP

Sistema de información de estudiantes



Requisitos

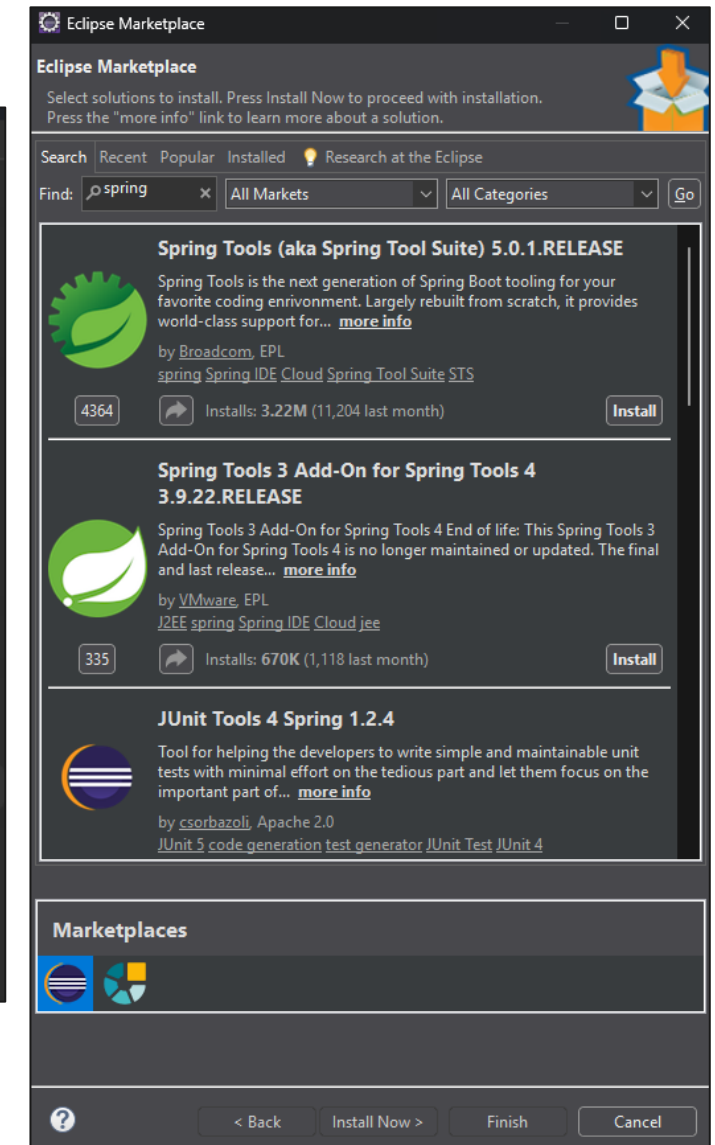
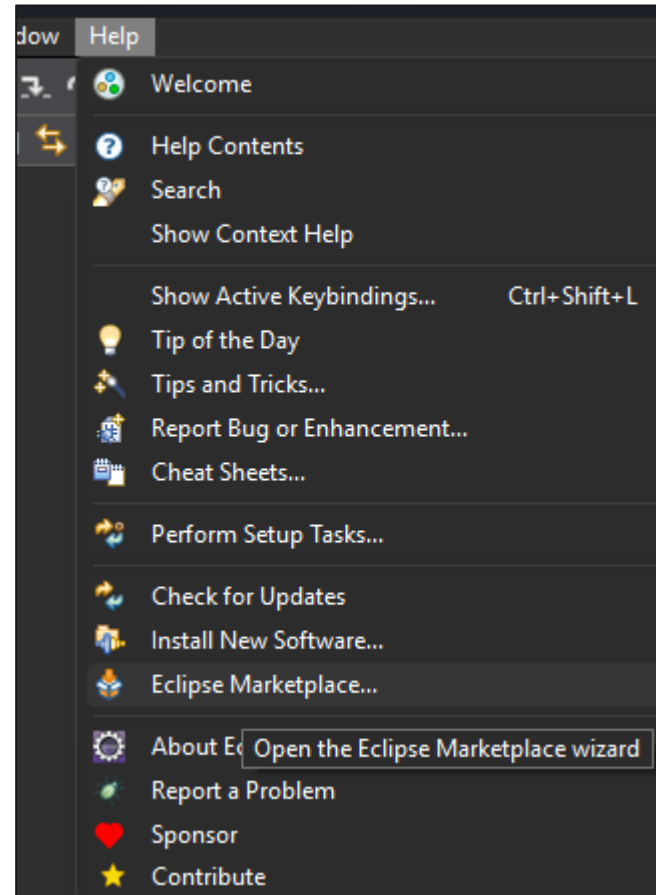
- Java JDK 8
- Eclipse IDE
- Spring Tools 4
- SoapUI



Instalaciones

Spring Tool Suite Instalación (1/2)

Para la instalación deben dirigirse al Marketplace de Eclipse que se encuentra en la pestaña de ayuda y luego buscar directamente Eclipse y seleccionar el que se llama : Spring Tools (aka Spring Tool Suite)



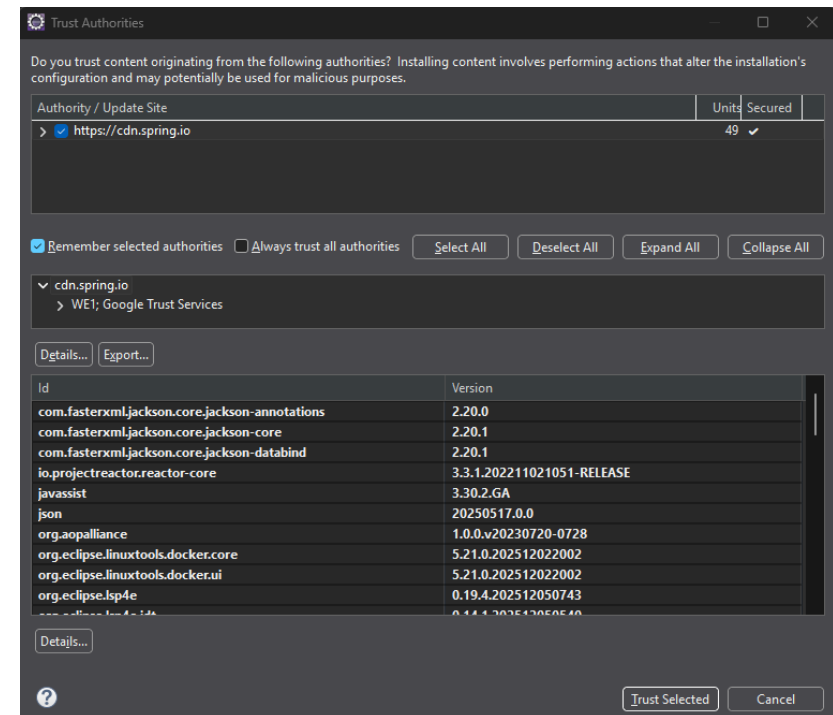
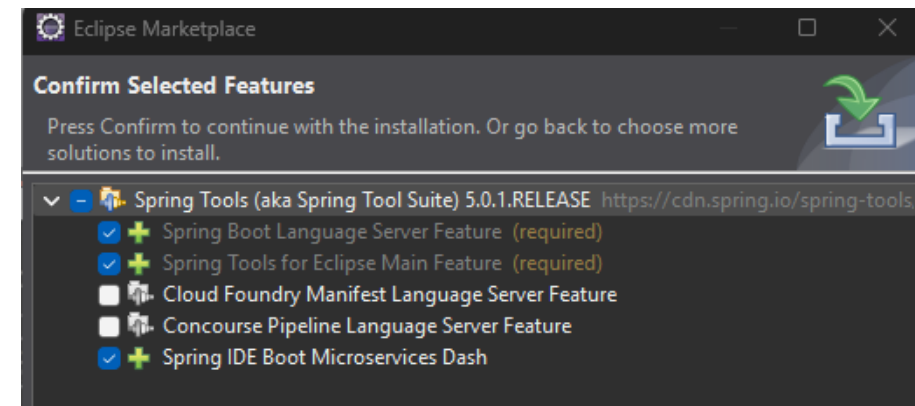
Instalaciones

Spring Tool Suite Instalación (2/2)

En la instalación solo deberán dejar todo por default y dar click en confirm y luego aceptar la licencia y dar click en finish.

En caso de que les salga una ventana de trust authorities , solo deben seleccionar el url de spring y darle click en trust selected.

Finalmente, solo deben cerrar y volver a abrir el programa.

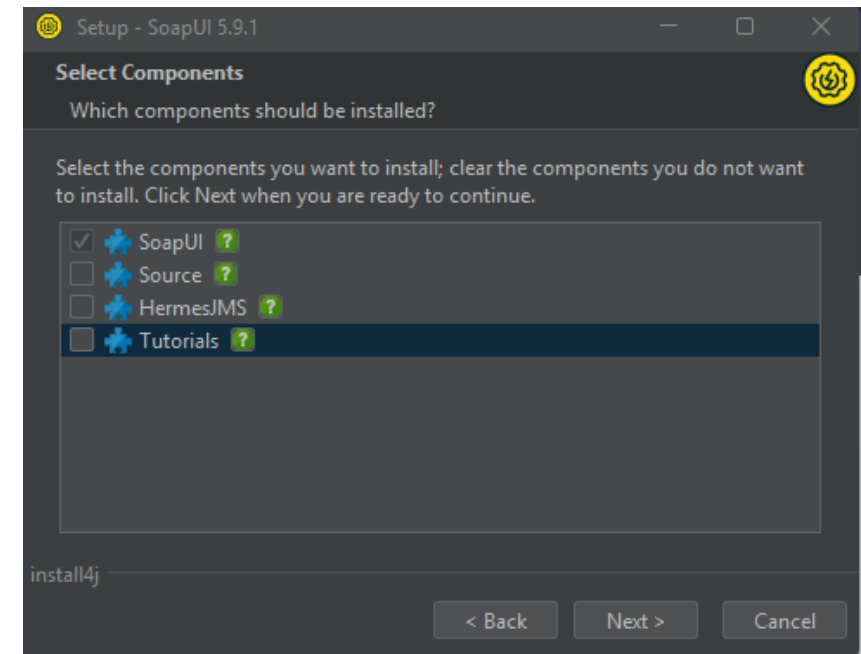
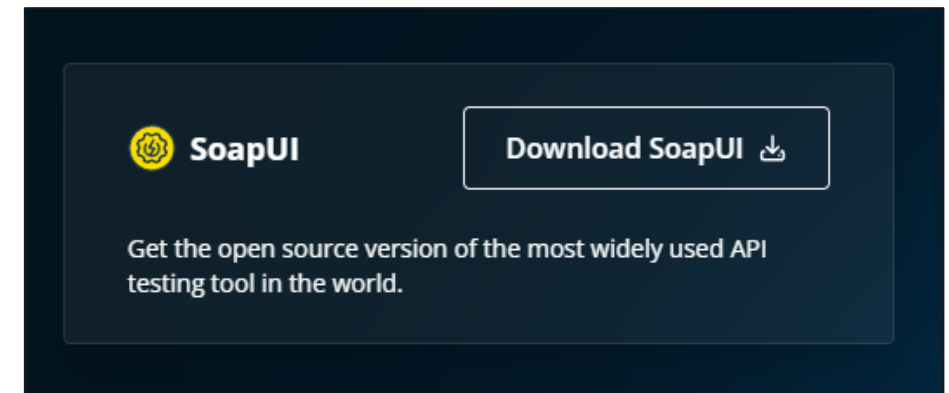


Instalaciones

SOAP UI Instalación

El archivo de instalación se encuentra en el GitHub y solamente tendrá que descargarlo desde el enlace

Una vez ejecuten el archivo de instalación solo deberán seleccionar en next en cada pantalla y seleccionar solo soap UI.



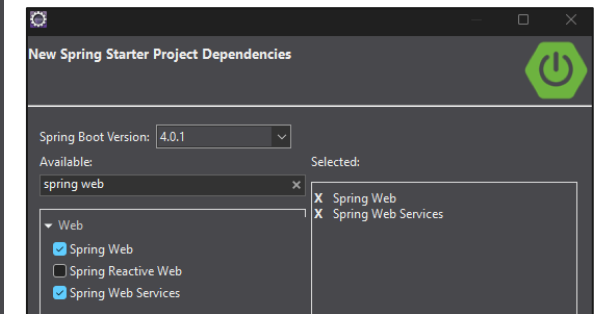
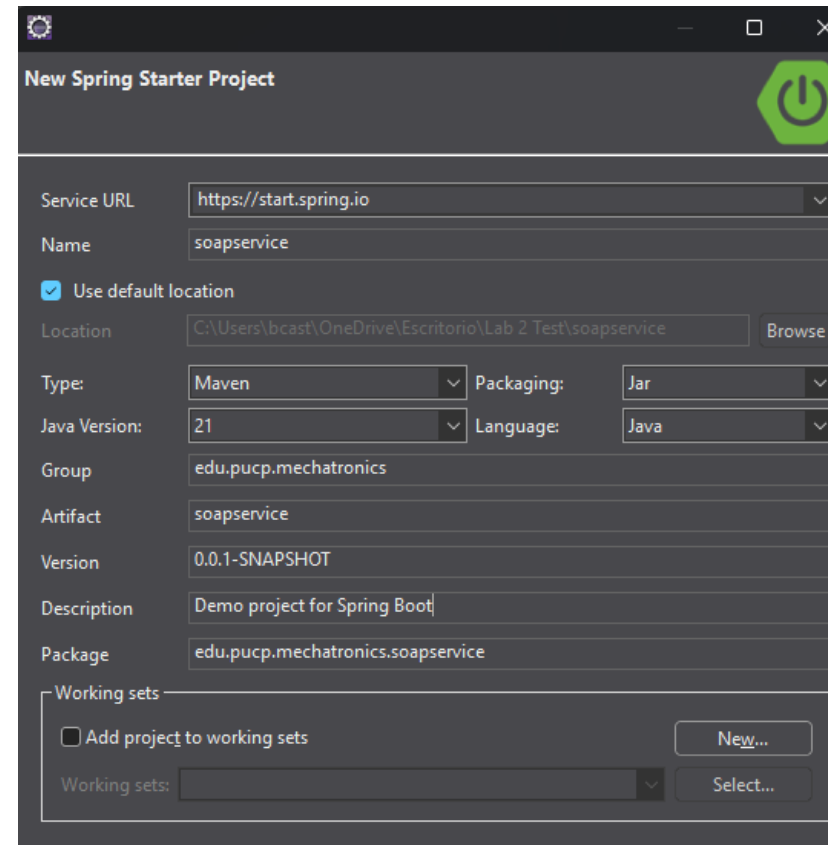
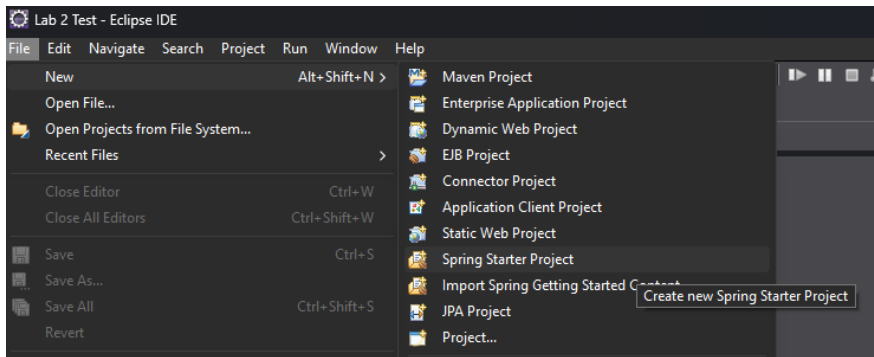
Proyecto

Configuración del servidor SOAP (1/3)

Para esto vamos a crear un nuevo proyecto, para ello nos vamos a File, New y luego a Spring Starter Project. En caso de que no aparezca, puede ser encontrado en el apartado Other en New.

Aquí llenaremos la data para el Project Metadata (soapservice), además el proyecto debe estar en Maven con lenguaje Java 21 y además llamaremos al Group como edu.pucp.mechatronics.

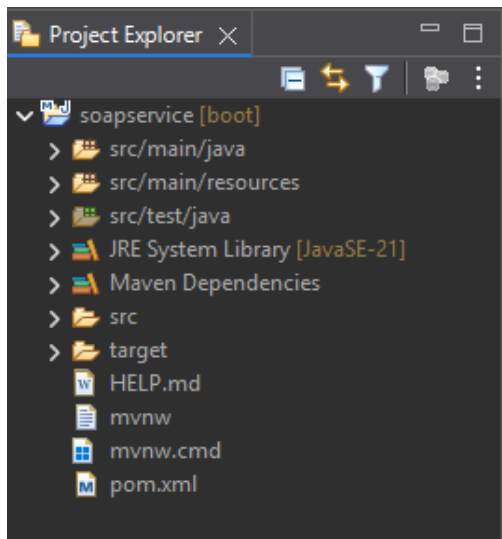
Así mismo, añadiremos Spring Web y Spring Web Services como dependencias.



Instalaciones

Configuración del servidor SOAP (2/3)

Una vez el proyecto haya terminado de cargar debería poderse ver de la siguiente forma.



Una vez ya se tenga todo cargado, debemos modificar el archivo POM del proyecto.

Aquí debemos añadir EL Toolikt Web Services Decription Language for Java (wsdl4j) y el jaxb-runtime

Luego para actualizar el Proyecto , grabamos con ctrl + s , luego, click derecho, maven, update project , OK

Instalaciones

Configuración del servidor SOAP (3/3)

Una vez ya se tenga todo cargado, debemos modificar el archivo POM del proyecto.

Aquí debemos añadir EL Toolikt Web Services Decription Language for Java (wsdl4j) y el jaxb-runtime en conjunto con el jakarta.xml

Luego para actualizar el Proyecto , grabamos con ctrl + s , luego, click derecho, maven, update project , OK

```
<dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
</dependency>
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>
```

Implementación del Proyecto SOAP

Se crearán los siguientes paquetes y archivos :

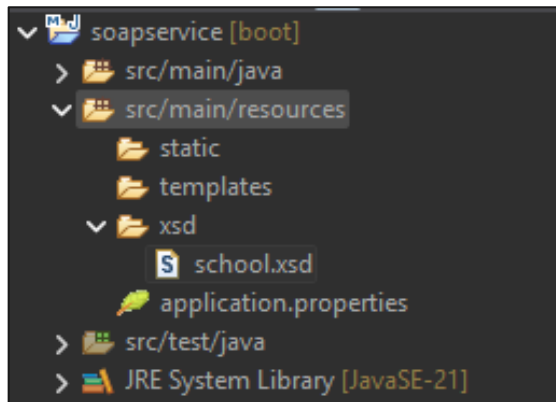
school.xsd : Define las clases y métodos del servicio

edu.pucp.mechatronics.soapservices :
Este contiene clases para publicar el wsdl
Contiene la data para el repositorio del servicio
Contiene la clase principal para que pueda correr la aplicación

pe.edu.pucp.xml.school :
Contiene las clases autogeneradas para el archivo xsd
Contiene la clase student
Contiene las clases para enviar y recibir request del servicio

Implementación del Proyecto SOAP

Implementación del archivo school.xsd :
Este va como recurso en el apartado src/main/resources, aquí debe crearse una carpeta xsd y ahí crear el archivo school.xsd donde se insertara el siguiente código:

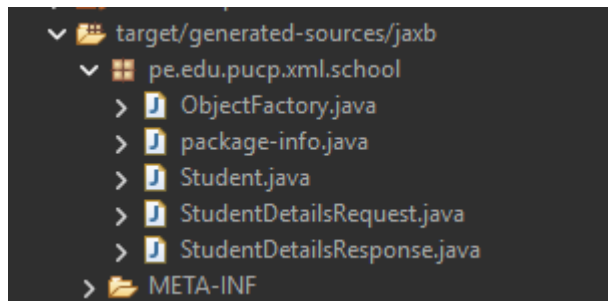


```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.pucp.edu.pe/xml/school"
  targetNamespace="http://www.pucp.edu.pe/xml/school"
  elementFormDefault="qualified">
  <xs:complexType name="Student">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="age" type="xs:int"/>
      <xs:element name="address" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="StudentDetailsRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="StudentDetailsResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Student" type="tns:Student"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Implementación del Proyecto SOAP

Añadiremos el plugin jaxb2 a nuestro archivo pom.xml para el generado automatico de clases desde schools.xsd.

Esto generara nuevos archivos localizados en la carpeta :
Target/generated/sources/jaxb



```
<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>
    <packageName>pe.edu.pucp.xml.school</packageName>
    <sources>
      <source>${project.basedir}/src/main/resources/xsd/school.xsd
    </source>
    </sources>
    <outputDirectory>${project.build.directory}/generated-
sources/jaxb</outputDirectory>
    <clearOutputDir>false</clearOutputDir>
  </configuration>
</plugin>
```

Implementación del Proyecto SOAP

Configuración del paquete edu.pucp.mechatronics.soapservice package
Este contiene clases para publicar el wsdl y contiene la data para el servicio del repositorio

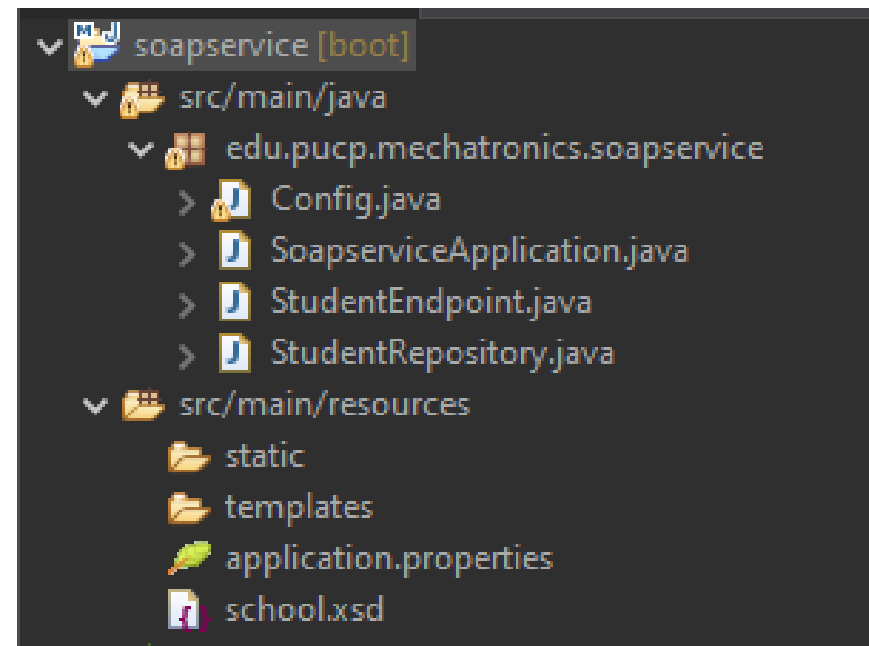
Este tendrá las clases :

Config

SoapserviceApplication

StudentRepository

StudentEndpoint



Config

```
package edu.pucp.mechatronics.soapservice;
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.WsConfigurer;
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 Config implements WsConfigurer {
    @Bean
    public ServletRegistrationBean<MessageDispatcherServlet>
messageDispatcherServlet(ApplicationContext applicationContext) {
        MessageDispatcherServlet servlet = new
```

```
MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean<>(servlet, "/service/*");
    }

    @Bean(name = "studentDetailsWsdl")
    public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema
schoolSchema) {
        DefaultWsdl11Definition wsdl11Definition = new
DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("StudentDetailsPort");
        wsdl11Definition.setLocationUri("/service");

        wsdl11Definition.setTargetNamespace("http://www.pucp.edu.pe/xml/schoo
");
        wsdl11Definition.setSchema(schoolSchema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema schoolSchema() {
        return new SimpleXsdSchema(new ClassPathResource("xsd/school.xsd"))
    }
}
```

SoapserviceApplication

```
package edu.pucp.mechatronics.soapservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SoapserviceApplication {

    public static void main(String[] args) {
        SpringApplication.run(SoapserviceApplication.class, args);
    }

}
```

StudentEndpoint

```
package edu.pucp.mechatronics.soapservice;

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 pe.edu.pucp.xml.school.StudentDetailsRequest;
import pe.edu.pucp.xml.school.StudentDetailsResponse;

@Endpoint
public class StudentEndpoint {
    private static final String NAMESPACE_URI = "http://www.pucp.edu.pe/xml/school";

    private final StudentRepository studentRepository;

    public StudentEndpoint(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }
    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "StudentDetailsRequest")
    @ResponsePayload
    public StudentDetailsResponse getStudent(@RequestPayload StudentDetailsRequest request) {
        StudentDetailsResponse response = new StudentDetailsResponse();
        response.setStudent(studentRepository.findStudent(request.getName()));

        return response;
    }
}
```

StudentRepository

```
package edu.pucp.mechatronics.soapservice;
import java.util.HashMap;
import java.util.Map;
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Component;
import org.springframework.util.Assert;
import pe.edu.pucp.xml.school.Student;
```

```
@Component
public class StudentRepository {
    private static final Map<String, Student>
students = new HashMap<>();
    @PostConstruct
    public void initData() {
        Student student = new Student();
        student.setName("Hugo");
        student.setAge(27);
        student.setAddress("Lima");
        students.put(student.getName(), student);
student = new Student();
        student.setName("Victor");
        student.setAge(28);
        student.setAddress("Nara");
        students.put(student.getName(), student);
```

```
student = new Student();
student.setName("Benjamin");
student.setAge(25);
student.setAddress("Delhi");
students.put(student.getName(), student);
```

```
student = new Student();
student.setName("Jorge");
student.setAge(24);
student.setAddress("Tokyo");
students.put(student.getName(), student);
```

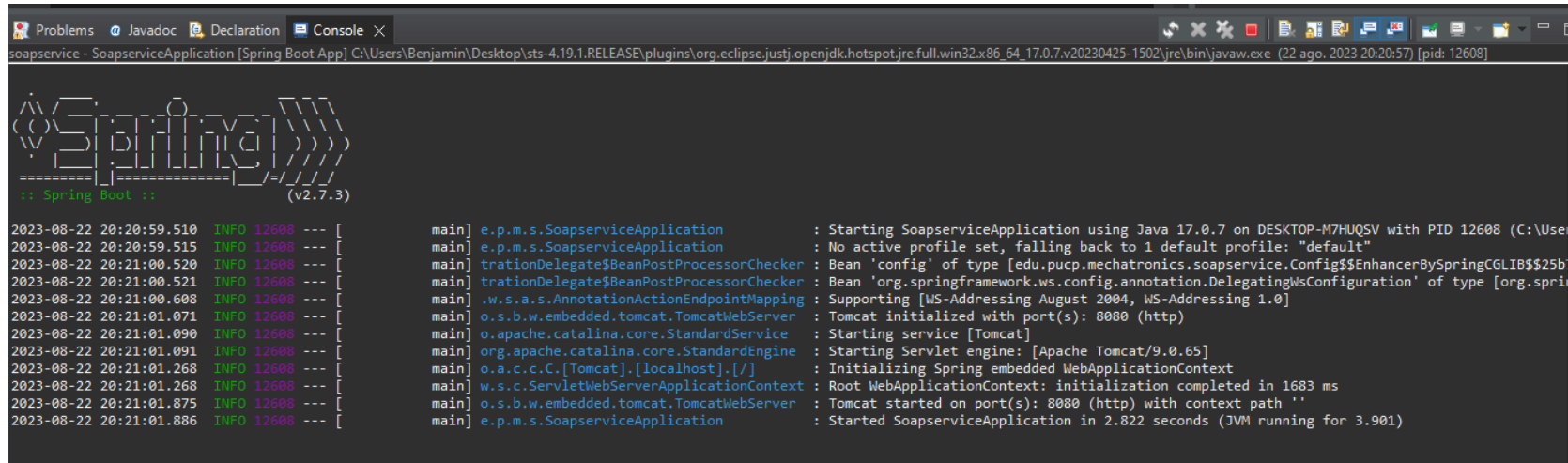
```
student = new Student();
student.setName("Felix");
student.setAge(26);
student.setAddress("Toronto");
students.put(student.getName(), student);
```

```
    }

    public Student findStudent(String name) {
        Assert.notNull(name, "The Student's name must not be
null");
        return students.get(name);
    }
}
```

Corriendo el Servidor (1/2)

Para poder correr el servidor es necesario dar click derecho en el proyecto, run as , spring boot app



```
soapservice - SoapserviceApplication [Spring Boot App] C:\Users\Benjamin\Desktop\sts-4.19.1.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.7.v20230425-1502\jre\bin\javaw.exe (22 ago, 2023 20:20:57) [pid: 12608]

:: Spring Boot ::
(v2.7.3)

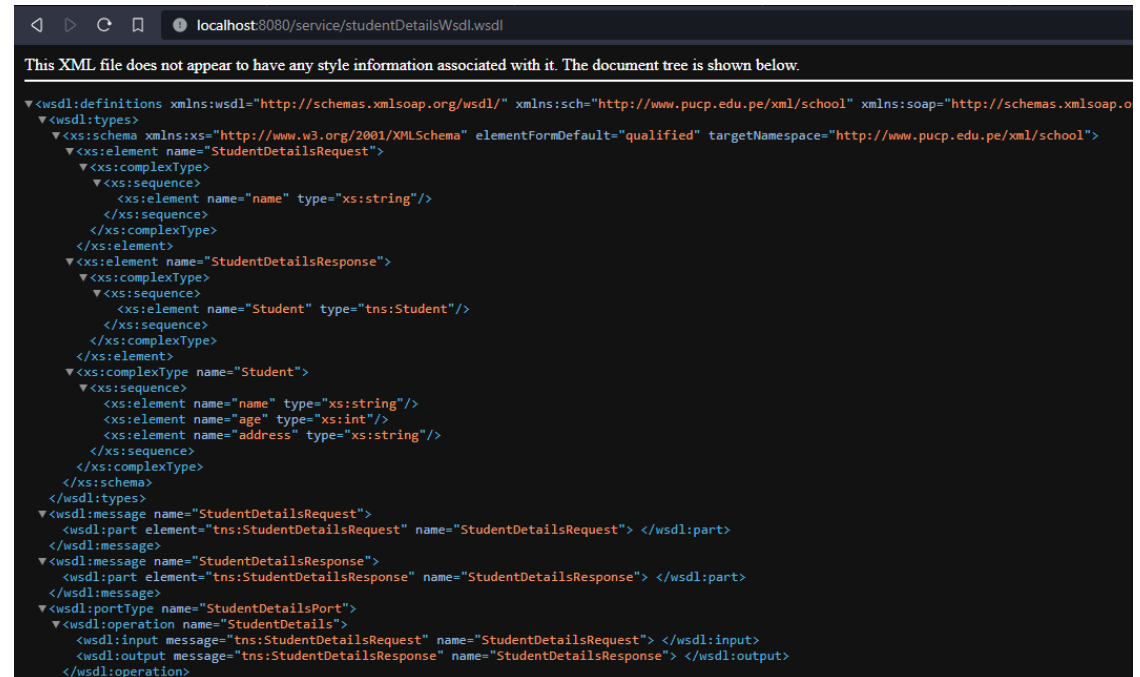
2023-08-22 20:20:59.510 INFO 12608 --- [main] e.p.m.s.SoapserviceApplication : Starting SoapserviceApplication using Java 17.0.7 on DESKTOP-M7HUQSV with PID 12608 (C:\Users\Benjamin\Desktop\sts-4.19.1.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.7.v20230425-1502\jre\bin\javaw.exe)
2023-08-22 20:20:59.515 INFO 12608 --- [main] e.p.m.s.SoapserviceApplication : No active profile set, falling back to 1 default profile: "default"
2023-08-22 20:21:00.520 INFO 12608 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'config' of type [edu.pucp.mechatronics.soapservice.Config$$EnhancerBySpringCGLIB$$25b1e1e1] is not eligible for bean validation: [org.springframework.beans.factory.config.ConfigurableBeanFactory$1$$ExceptionWithSource$1]
2023-08-22 20:21:00.521 INFO 12608 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotation.DelegatingWsConfiguration' of type [org.springframework.ws.config.annotation.DelegatingWsConfiguration] is not eligible for bean validation: [org.springframework.beans.factory.config.ConfigurableBeanFactory$1$$ExceptionWithSource$1]
2023-08-22 20:21:00.608 INFO 12608 --- [main] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
2023-08-22 20:21:01.071 INFO 12608 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-08-22 20:21:01.090 INFO 12608 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-08-22 20:21:01.091 INFO 12608 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2023-08-22 20:21:01.268 INFO 12608 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-08-22 20:21:01.268 INFO 12608 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1683 ms
2023-08-22 20:21:01.875 INFO 12608 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-08-22 20:21:01.886 INFO 12608 --- [main] e.p.m.s.SoapserviceApplication : Started SoapserviceApplication in 2.822 seconds (JVM running for 3.901)
```

Corriendo el Servidor (2/2)

Se puede ver que el servidor esta corriendo correctamente en este link :

<http://localhost:8080/service/studentDetailsWsdL.wsdl>

Aquí se podrá ver el contenido del archivo .wsdl



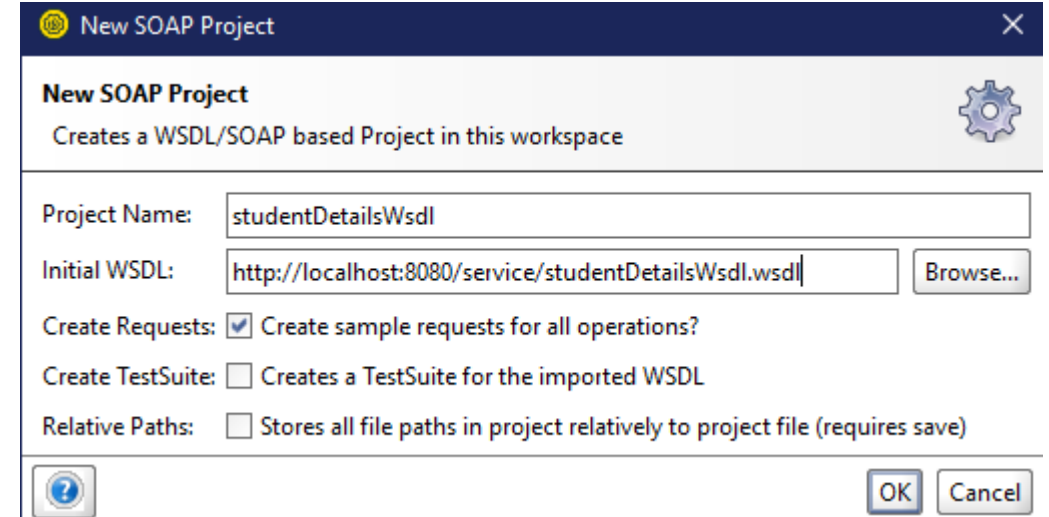
```
<?xml version='1.0' encoding='UTF-8'>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch="http://www.pucp.edu.pe/xml/school" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" targetNamespace="http://www.pucp.edu.pe/xml/school">
      <xs:element name="StudentDetailsRequest">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="StudentDetailsResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Student" type="tns:Student"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="Student">
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="age" type="xs:int"/>
          <xs:element name="address" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="StudentDetailsRequest">
    <wsdl:part element="tns:StudentDetailsRequest" name="StudentDetailsRequest"/>
  </wsdl:message>
  <wsdl:message name="StudentDetailsResponse">
    <wsdl:part element="tns:StudentDetailsResponse" name="StudentDetailsResponse"/>
  </wsdl:message>
  <wsdl:portType name="StudentDetailsPort">
    <wsdl:operation name="StudentDetails">
      <wsdl:input message="tns:StudentDetailsRequest" name="StudentDetailsRequest"/>
      <wsdl:output message="tns:StudentDetailsResponse" name="StudentDetailsResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```


Probando el proyecto (1/2)

Para hacer las pruebas usaremos SOAP UI , Para esto daremos click en File, new soap Project , luego en Initial WSDL , añadiremos el url.

<http://localhost:8080/service/studentDetailsWSDL.wsdl>

Aquí el nombre del proyecto se generará automáticamente.

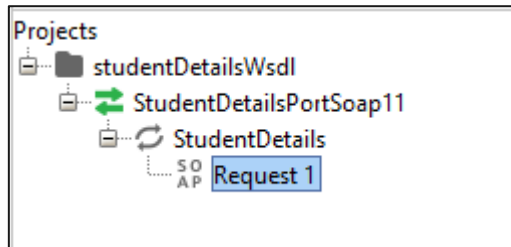


Probando el proyecto (2/2)

Esto nos generará nuestro nuevo proyecto, y una vez este hecho debemos hacer click en Request 1

Esto abrirá un nuevo request , que nos permitirá añadir un estudiante, para eso ingresaremos un nombre que este en StudentRepository entre <sch:name> y enviaremos el request

Esto nos dará como resultado la información de uno de los estudiantes registrados dentro de StudentRepository.java



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <sch:StudentDetailsRequest>
      <sch:name>Felix</sch:name>
    </sch:StudentDetailsRequest>
  </soapenv:Body>
</soapenv:Envelope>
```



```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:StudentDetailsResponse xmlns:ns2="http://www.pucp.edu.pe/xml/school">
      <ns2:Student>
        <ns2:name>Felix</ns2:name>
        <ns2:age>26</ns2:age>
        <ns2:address>Toronto</ns2:address>
      </ns2:Student>
    </ns2:StudentDetailsResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Desarrollo de nuestra aplicación REST

Sistema de Clientes para un negocio



Metodos a utilizar

HTTP METHODS

GET

POST

PUT

Delete

Get Customers

Request

GET /api/customers

Response

```
[  
  { id: 1, name: 'Hugo'},  
  { id: 2, name: 'Kim'},  
  ...  
]
```

Metodos a utilizar

HTTP METHODS

GET

POST

PUT

Delete

Get a Customer

Request

GET /api/customers/1

Response

{ id: 1, name: 'Hugo' }

Metodos a utilizar

HTTP METHODS

GET

POST

PUT

Delete

Create a Customer

Request

POST /api/customers

{ name: 'Kim' }

Response

{ id: 1, name: 'Kim' }

Metodos a utilizar

HTTP METHODS

GET

POST

PUT

Delete

Update a Customer

Request

PUT /api/customers/1

{ name: 'Fernando' }

Response

{ id: 1, name: 'Fernando' }

Metodos a utilizar

HTTP METHODS

GET

POST

PUT

DELETE

Delete a Customer

Request **DELETE** `/api/customers/1`

Response

`{ id: 1, name: 'Fernando' }`

Requisitos

- Node JS
- npm
- Express JS



Node JS

Para su instalación solo deben descargar la versión LTS más reciente desde el link provisto en GitHub y darle click en siguiente en cada apartado

En caso de que quieran verificar la instalación pueden usar el comando CMD : `node --version`

Download Node.js®

Get Node.js® v24.13.0 (LTS) for Windows using Docker with npm

Info Want new features sooner? Get the [latest Node.js version](#) instead and try the latest improvements!

```
1 # Docker has specific installation instructions for each operating system.
2 # Please refer to the official documentation at https://docker.com/get-started/
3
4 # Pull the Node.js Docker image:
5 docker pull node:24-alpine
6
7 # Create a Node.js container and start a Shell session:
8 docker run -it --rm --entrypoint sh node:24-alpine
9
10 # Verify the Node.js version:
11 node -v # Should print "v24.13.0".
12
13 # Verify npm version:
14 npm -v # Should print "11.6.2".
```

PowerShell </> Copy to clipboard

Docker is a containerization platform. If you encounter any issues please visit [Docker's website](#)

Or get a prebuilt Node.js® for Windows running a x64 architecture.

Windows Installer (.msi) Standalone Binary (.zip)

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.7623]
(c) Microsoft Corporation. Todos los derechos reservados.

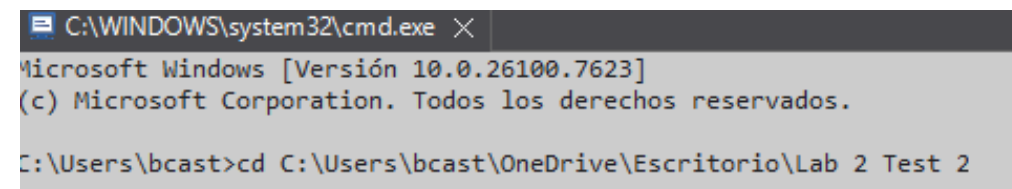
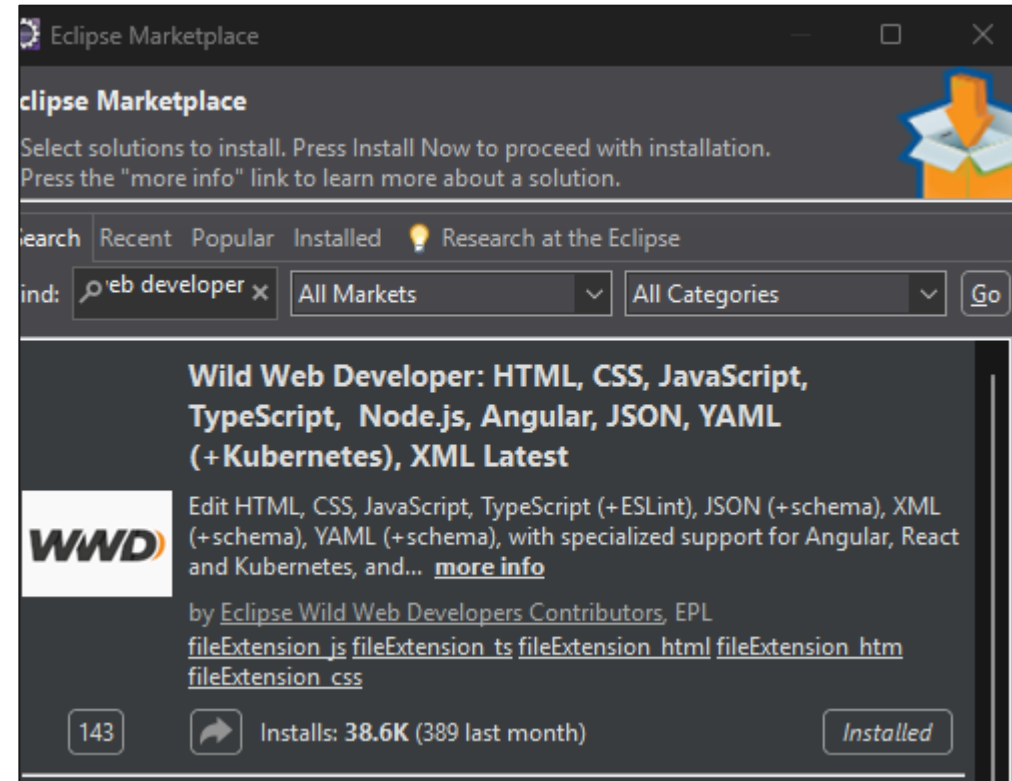
C:\Users\bcast>node --version
v24.13.0
```

NPM

NPM Instalación

Para la instalación deben dirigirse al Marketplace de Eclipse que se encuentra en la pestaña de ayuda y luego buscar directamente Eclipse y seleccionar el que se llama : Wild web developer y luego reiniciar eclipse.

Una vez realizado esto en la consola debemos dirigirnos directamente a la carpeta donde trabajaremos con el comando CD

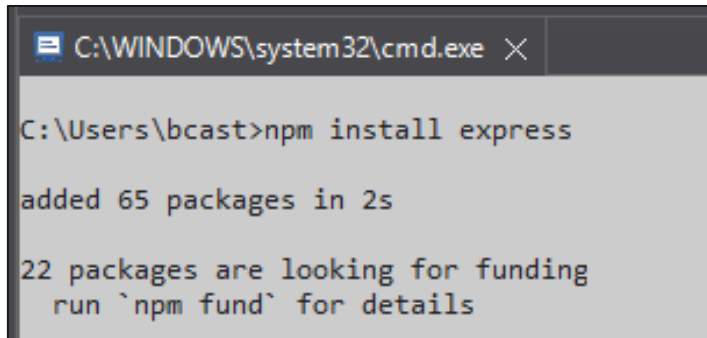


Configuración del proyecto

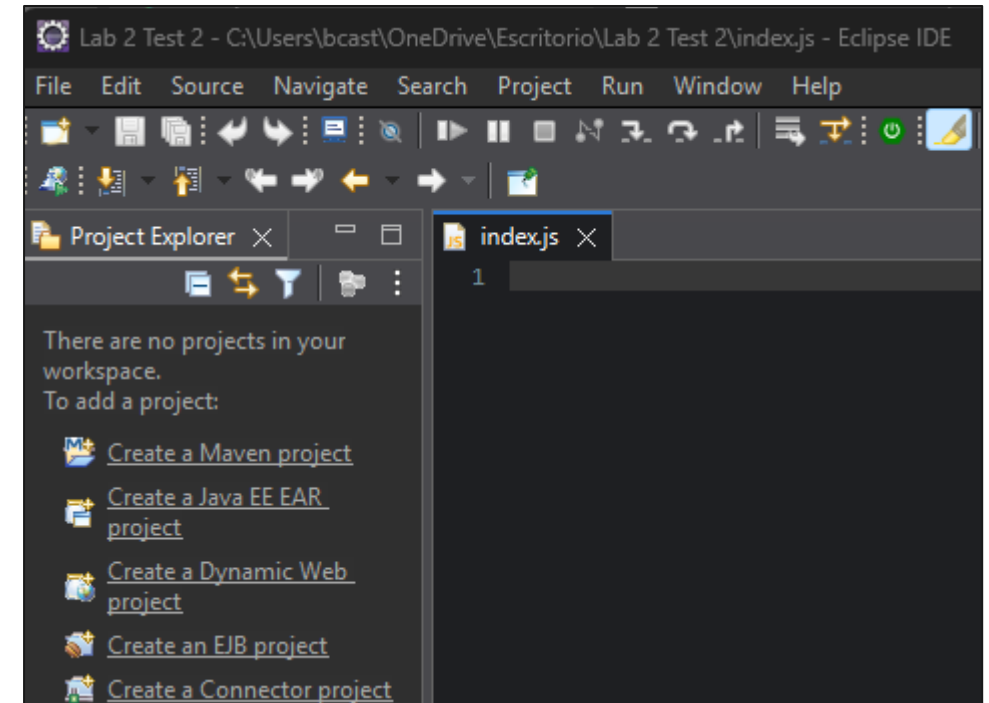
Para la configuración del proyecto utilizaremos un framework, en este caso Express JS , para su instalación en nuestro proyecto solo debemos escribir el comando `npm install express` en el terminal una vez ya estemos en la carpeta donde vamos a trabajar.

Una vez instalado haremos un nuevo archivo `index.js` con el siguiente código :
`type nul > index.js`

Para abrirlo simplemente nos vamos a File , Open File y lo buscamos.



```
C:\WINDOWS\system32\cmd.exe X
C:\Users\bcast>npm install express
added 65 packages in 2s
22 packages are looking for funding
run `npm fund` for details
```



Configuración del proyecto

Una vez abierto insertaremos el siguiente código y lo guardaremos. Finalmente solo debemos escribir en la consola node index.js para ejecutarlo.



The screenshot shows the Eclipse IDE with a file named 'index.js' open. The code in the editor is as follows:

```
1 const express = require ('express')
2 const app = express()
3
4 app.use(express.json())
5
6 const customers = [
7   {id :1 , name: "Hugo"},
8   {id: 2 , name: "Kim"}
9 ]
10
11 app.get('/',(req,res)=>{
12   res.send('Hola mundo');
13 })
14
15 app.get('/api/customers', (req,res)=> {
16   res.send(customers);
17 })
18
19 app.listen(3000,
20   ()=> console.log('Escuchando en el puerto 3000....'));
21
```

Below the editor, the 'Terminal' tab is active, showing the command prompt with the command 'node index.js' and its output 'Escuchando en el puerto 3000....'.

```
const express = require ('express')
const app = express()
```

```
app.use(express.json())
```

```
const customers = [
  {id :1 , name: "Hugo"},
  {id: 2 , name: "Kim"}
]
```

```
app.get('/',(req,res)=>{
  res.send('Hola mundo');
})
```

```
app.get('/api/customers', (req,res)=> {
  res.send(customers);
})
```

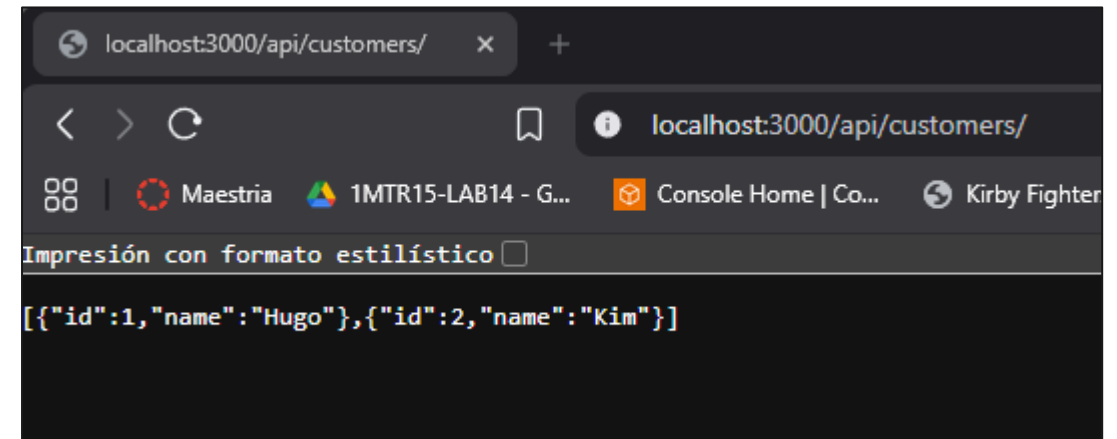
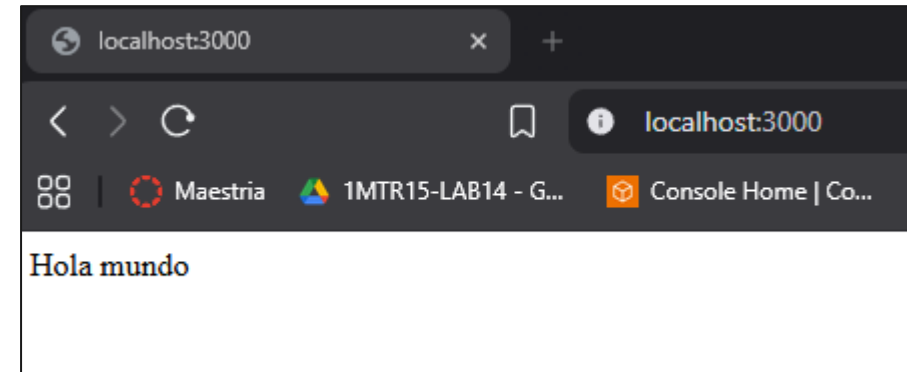
```
app.listen(3000,
  ()=> console.log('Escuchando en el puerto 3000....'));
```

Configuración del proyecto

Para probar que funciona , podemos simplemente darle a <http://localhost:3000/> para probar el GET

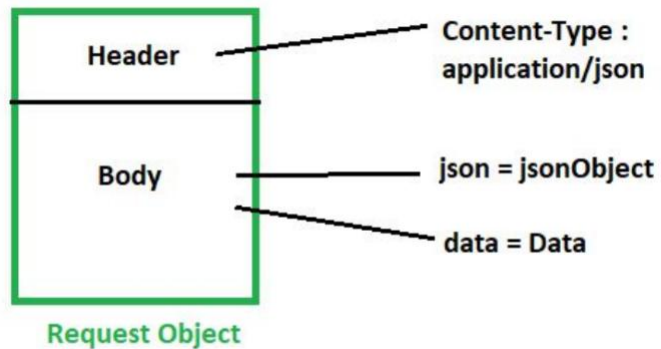
O

<http://localhost:3000/api/customers/> para probar la información insertada.



Configuración del proyecto

Continuando, implementaremos el método post , el cual consta de un Header y un body para solicitar el objeto a trabajar



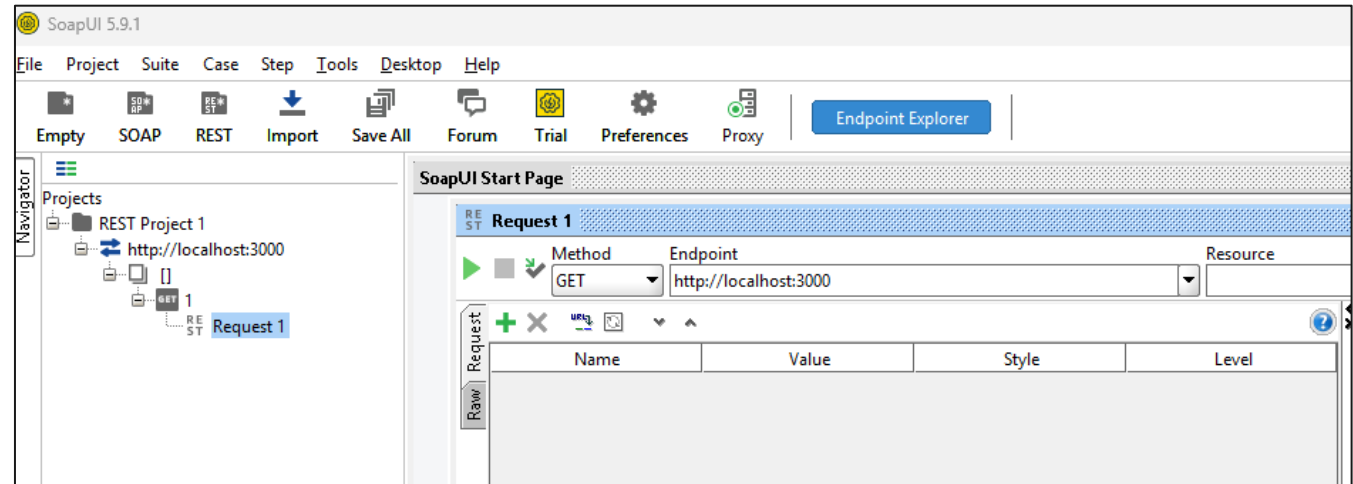
```
app.get('/api/customers/:id', (req,res)=> {  
  const customer = customers.find(c => c.id ===parseInt(req.params.id));  
  if(!customer){  
    res.status(404).send('El usuario con este ID no se encuentra')  
  }  
  res.send(customer)  
})
```

```
app.post ('/api/customers', (req,res)=> {  
  const customer = {  
    id: customers.length + 1,  
    name : req.body.name,  
  }  
  customers.push(customer);  
  res.send(customer);  
}  
)
```

Probando el proyecto

Con el fin de poder probar lo que hemos implementado hasta ahora, haremos uso de Soap UI para ello abriremos el programa, nos iremos al apartado de File y seleccionaremos New Rest Project

Aquí insertaremos nuestra URL Principal
<http://localhost:3000>



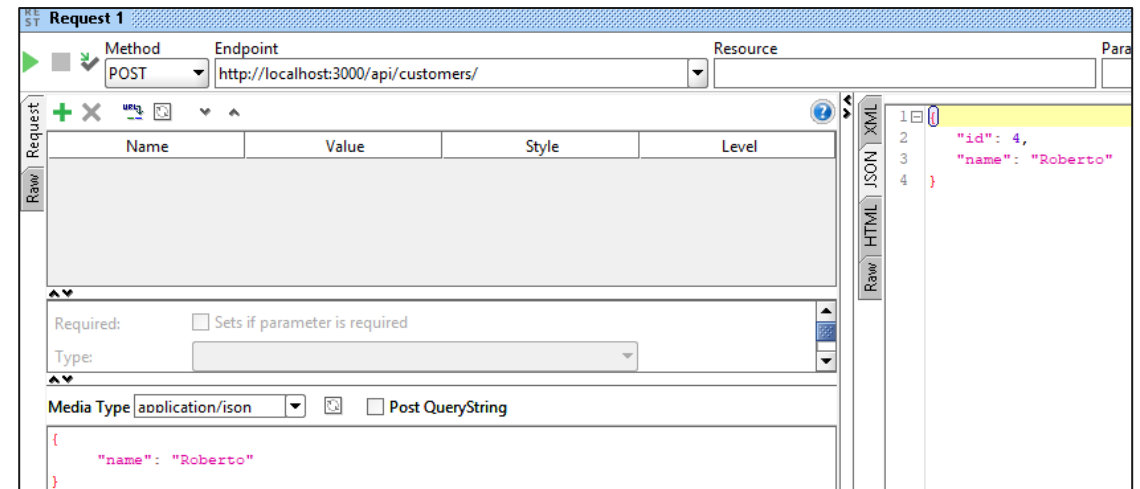
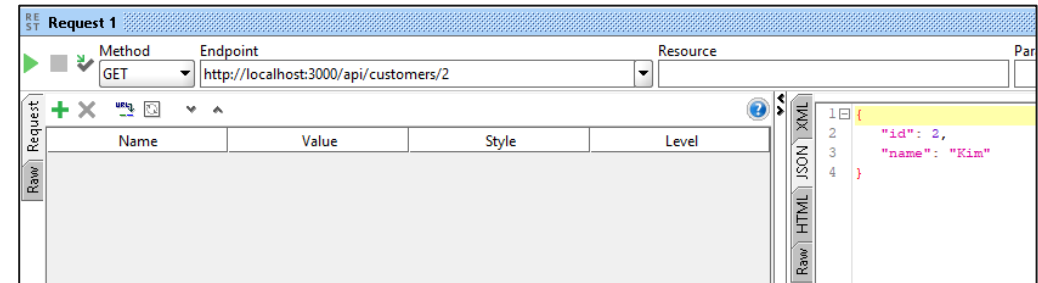
Probando el proyecto

Una vez tengamos esto , haremos la prueba con los apis creados, para esto solo debemos seleccionar el método correspondiente e ir cambiando la URL y luego darle click a Play

Aquí podremos probar usando el link:
<http://localhost:3000> para probar el entorno GET

Tambien podemos probar usando :
<http://localhost:3000/api/customers/ID>

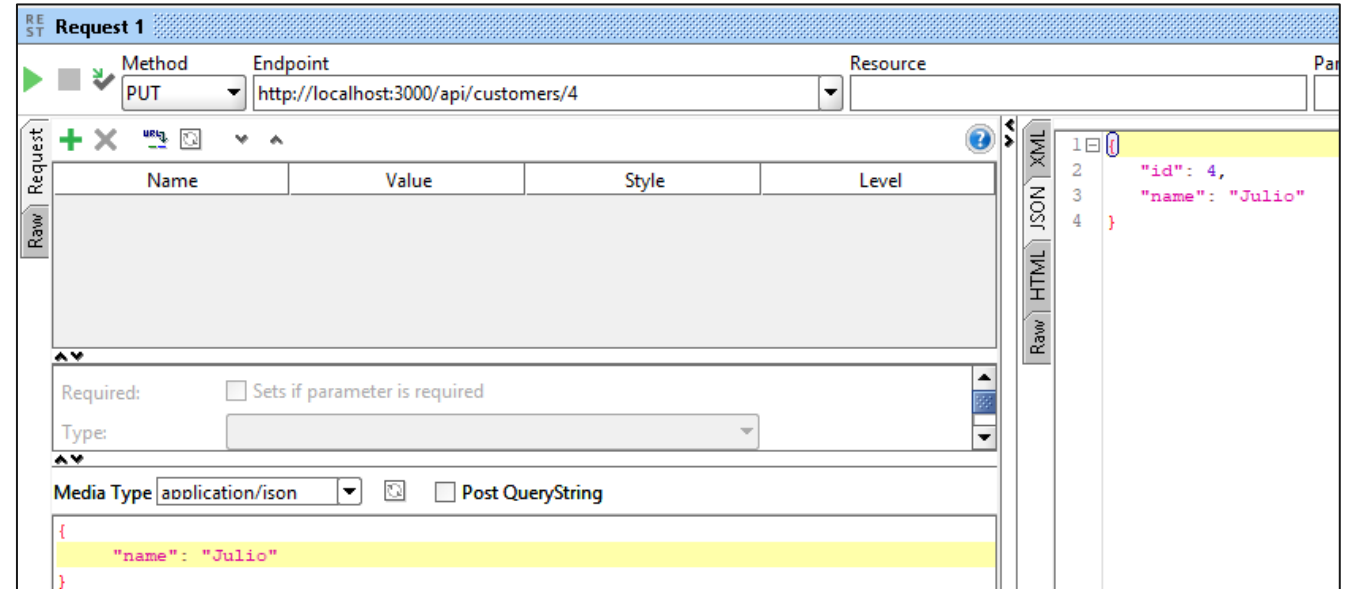
Y finalmente podemos probar usando el método POST
en : <http://localhost:3000/api/customers> e insertando los headers correspondientes, en este caso el name



Probando el proyecto

Continuando, tambien implementaremos el método put

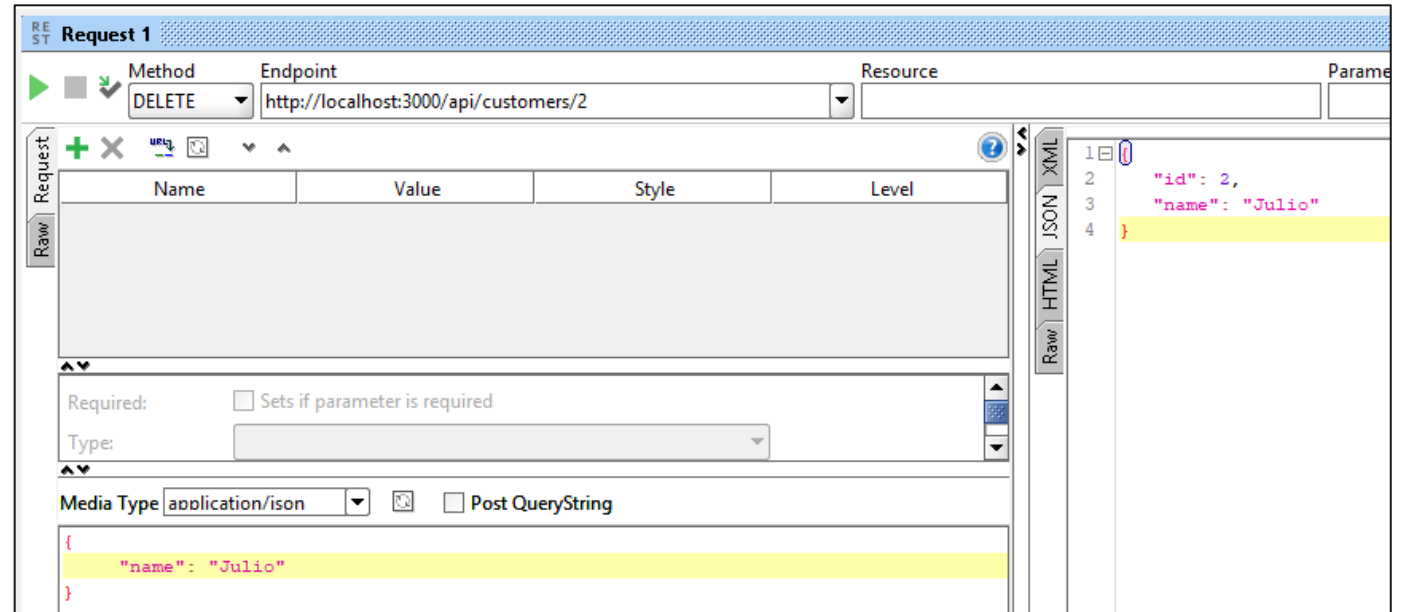
```
app.put('/api/customers/:id', (req,res)=> {  
  const customer = customers.find(c => c.id  
  ===parseInt(req.params.id));  
  if(!customer){  
    res.status(404).send('El usuario con este  
ID no se encuentra')  
  }  
  if(!req.body.name ||  
req.body.name.length<3){  
    res.status(403).send("El nombre debe  
tener al menos 3 caracteres")  
    return  
  }  
  
  customer.name =req.body.name;  
  res.send(customer);  
})
```



Probando el proyecto

Finalmente implementaremos el método delete

```
app.delete('/api/customers/:id', (req,res)=>
{
  const customer = customers.find(c =>
c.id ===parseInt(req.params.id));
  if(!customer){
    res.status(404).send('El usuario con
este ID no se encuentra')
  }
  const index =
customers.indexOf(customer);
customers.splice(index,1);
res.send(customer);
})
```



Información importante

Swagger – Open Api <https://petstore.swagger.io/>

pet Everything about your Pets Find out more: <http://swagger.io>

GET /pet/{petId} Find pet by ID

POST /pet/{petId} Updates a pet in the store with form data

DELETE /pet/{petId} Deletes a pet

POST /pet/{petId}/uploadImage uploads an image

POST /pet Add a new pet to the store

PUT /pet Update an existing pet

GET /pet/findByStatus Finds Pets by status

GET /pet/findByStatus Finds Pets by status

Multiple status values can be provided with comma separated strings

Parameters Cancel

Name	Description
status *required array[string] (query)	Status values that need to be considered for filter

available
pending
sold

Execute **Clear**

Responses Response content type: application/json

Curl

```
curl -X 'GET' \
  'https://petstore.swagger.io/v2/pet/findByStatus?status=available' \
  -H 'accept: application/json'
```

Request 1

Method: GET Endpoint: <https://petstore.swagger.io/v2/swagger.json> Resource: Parameters:

Raw **HTML** **JSON** **XML**

```
1 {
2   "swagger": "2.0",
3   "info": {
4     "description": "This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io
5     "version": "1.0.7",
6     "title": "Swagger Petstore",
7     "termsOfService": "http://swagger.io/terms/",
8     "contact": {"email": "apiteam@swagger.io"},
9     "license": {
10      "name": "Apache 2.0",
11      "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
12    }
13  },
14  "host": "petstore.swagger.io",
15  "basePath": "/v2",
16  "tags": [
17    {
18      "name": "pet",
19      "description": "Everything about your Pets",
20      "externalDocs": {
21        "description": "Find out more",
22        "url": "http://swagger.io"
23      }
24    },
25    {
26      "name": "store",
27      "description": "Access to Petstore orders"
28    },
29    {
30      "name": "user",
31      "description": "Operations about user",
32      "externalDocs": {
33        "description": "Find out more about our store",
34        "url": "http://swagger.io"
35      }
36    }
37  ],
38  "schemes": [
39    "https",
40    "http"
41  ]
42 }
```