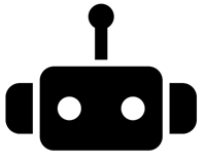

PUCP

SESIÓN DE LABORATORIO 2
Soap y Rest



HORARIO 10M1

Empezaremos a las 7:10 p.m

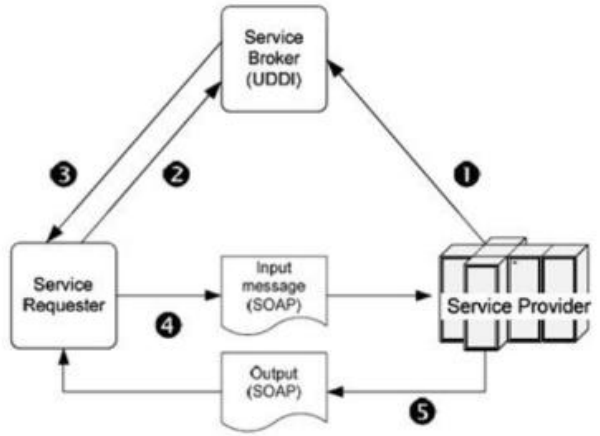
Gracias!

PUCP

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



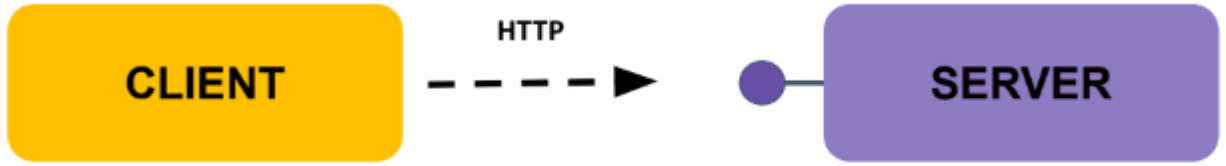
Steps

1. Publish the service (WSDL)
2. Consult the directory (UUDI)
3. Find the service (WSDL)
4. SOAP Request
5. SOAP Response

Elements

UUDI	Directory of the service
WSDL	Service description (XML)
XML	Data format
HTTP	Transfer protocol of the messages

Que es REST?



Representational **S**tate **T**ransfer

Operaciones CRUD

CRUD Operations

REST VS SOAP

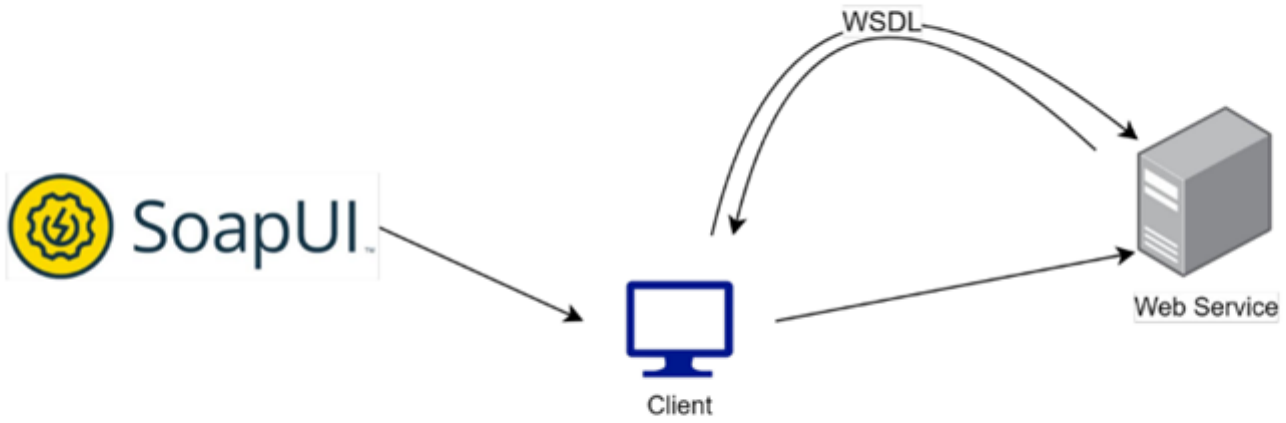
REST

- Social Media
- Web Chat
- Mobile

SOAP

- Financial
- Telecommunication
- Payment Gateways

Desarrollo de nuestra aplicación SOAP
Sistema de información de estudiantes



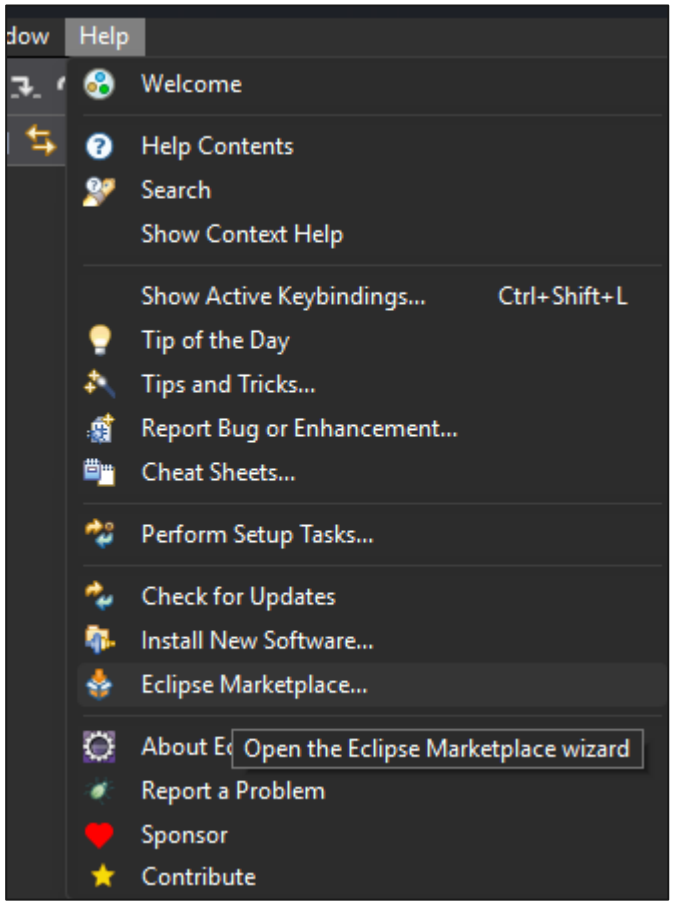
Requisitos

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

Instalaciones

Spring Tool Suite Instalación (1/2)

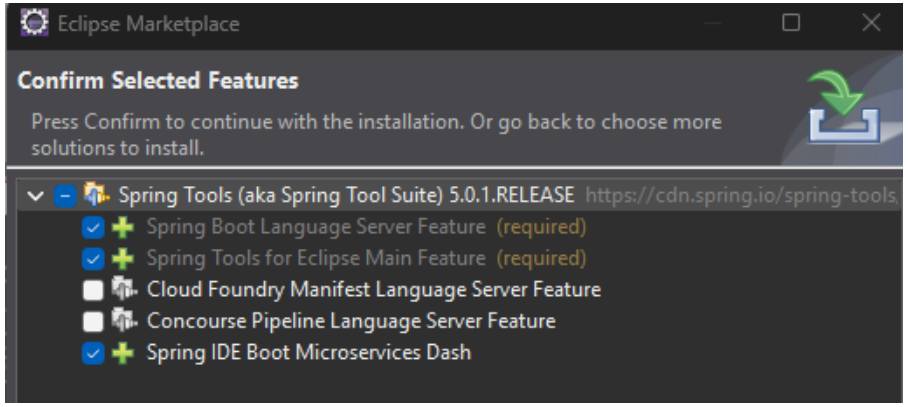
Para la instalación deben dirigirse al Marketplace de Eclipse que se encuentra en la pestaña de ayuda y luego buscar directamente Eclipse y seleccionar el que se llama : Spring Tools (aka Spring Tool Suite)

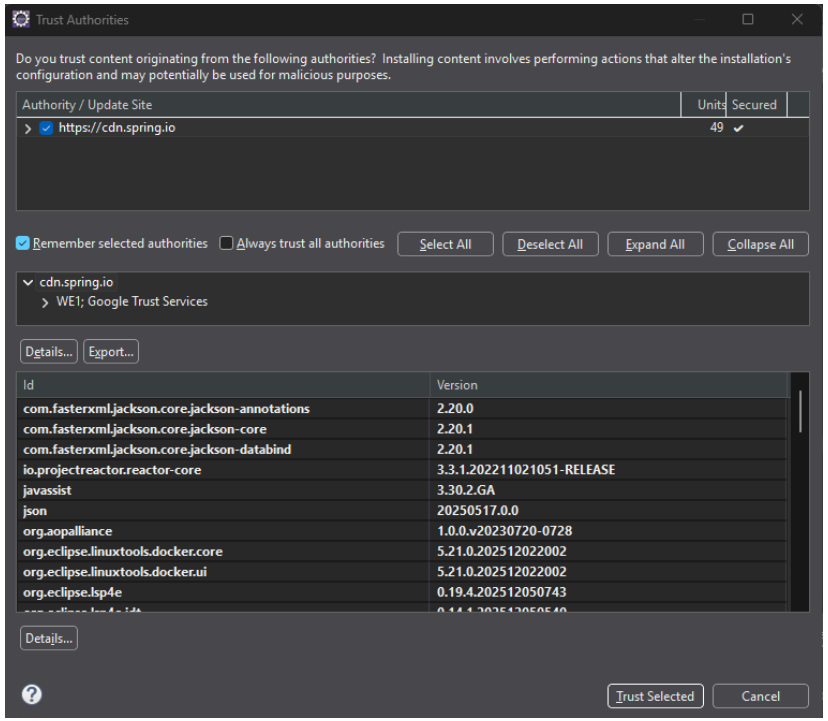


Instalaciones

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.

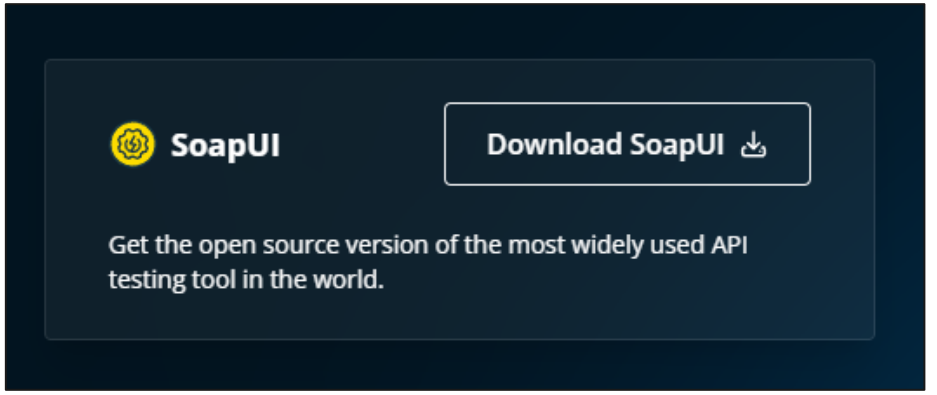


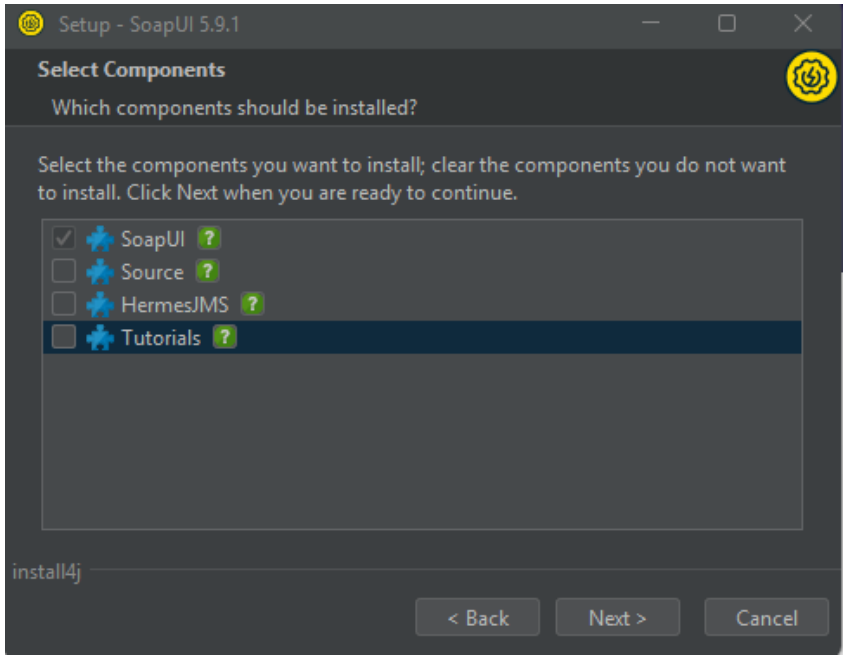


Instalaciones

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.





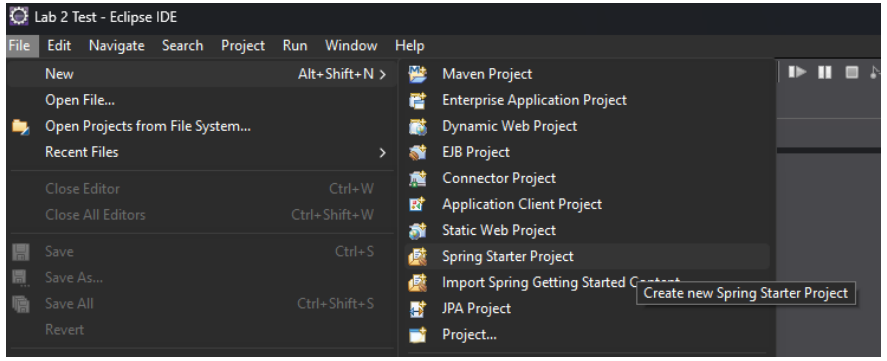
Proyecto

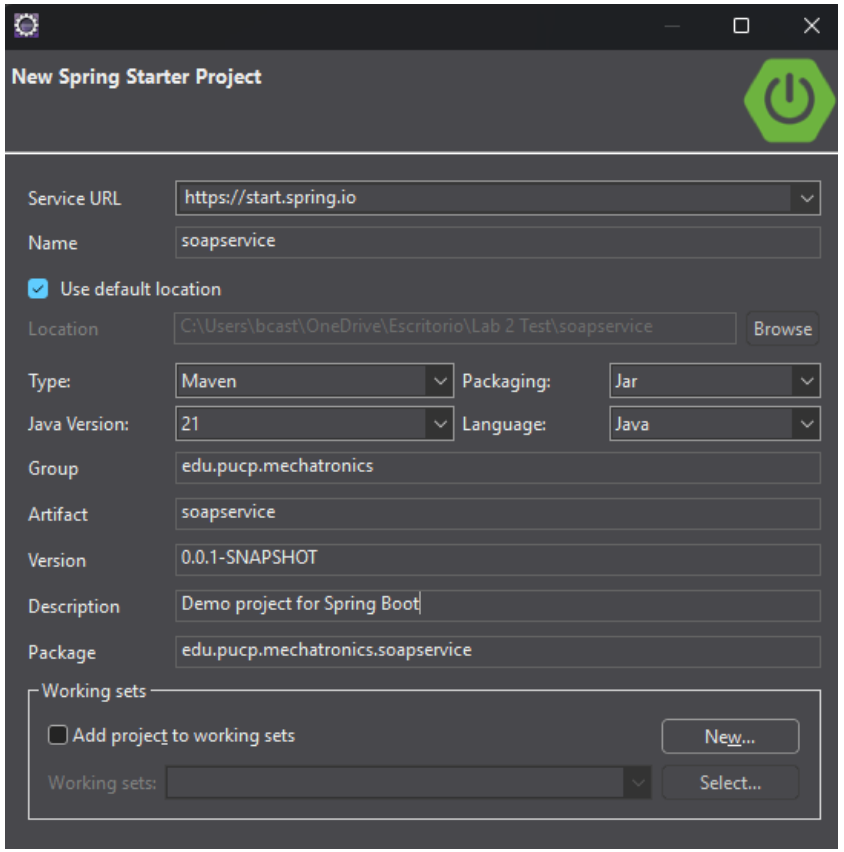
Configuracion del servidor SOAP (1/3)

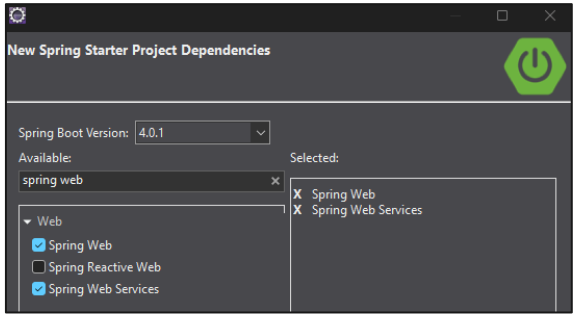
Para esto vamos a crear un nuevo proyecto, para ello nos vamos a file, New y luego a Spring Starter Project.
En caso de que no aparezca , puede ser encontrado en el apartado Other en New.

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.

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



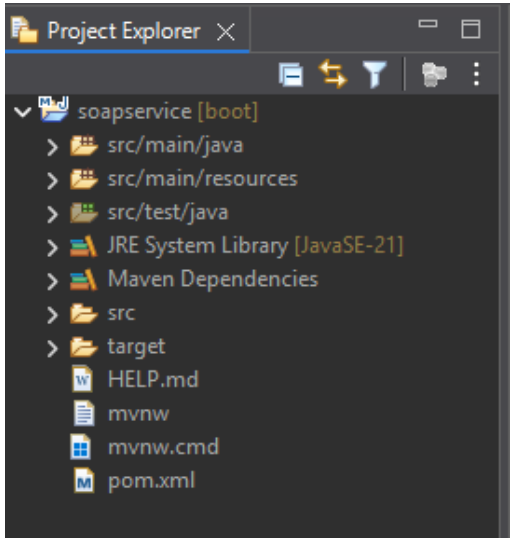




Instalaciones

Configuracion del servidor SOAP (2/3)

Una vez el proyecto haya terminado de cargar deberia 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

Configuracion 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 :

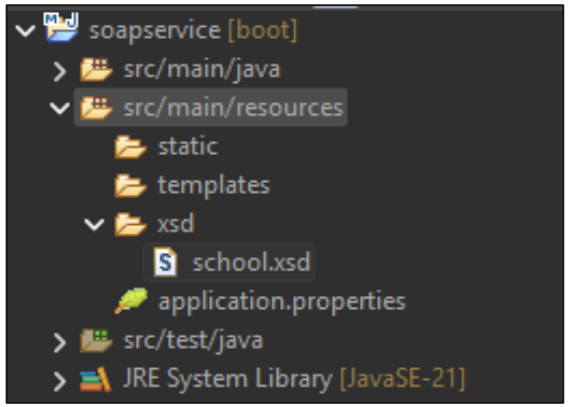
school.xsd : Define las clases y métodos del servicio

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

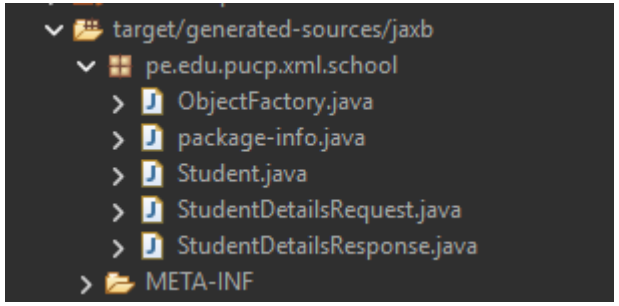
Implementacion del archivo school.xsd :
Este va como recurso en el apartado src/main/resources, aquí debe crearse una carpeta xsd y ahí crear el archivo school.xsd donde se insertara el siguiente codigo:




```xml
<?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



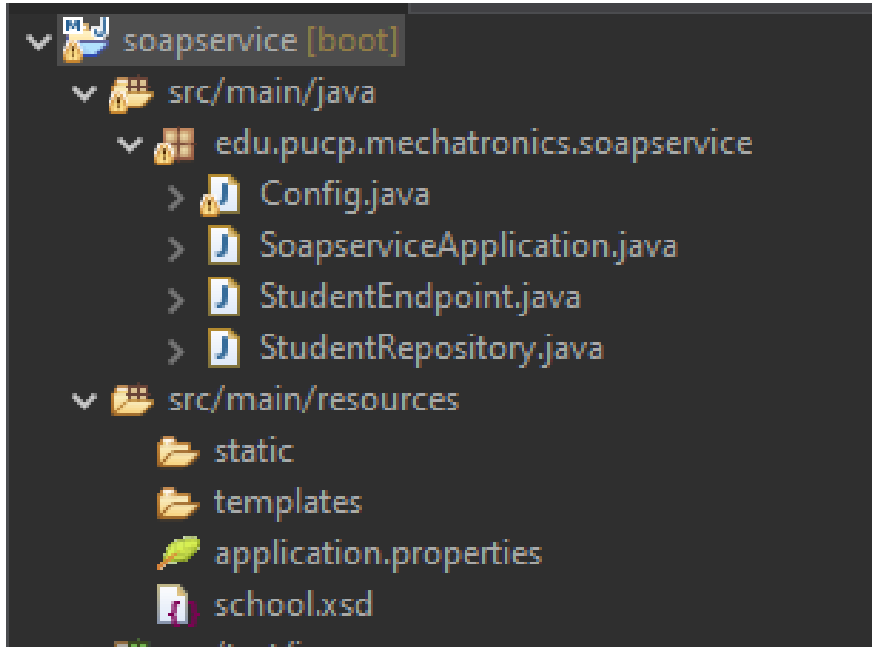

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxb2-maven-plugin</artifactId>
    <version>3.1.0</version>
    <executions>
        <execution>
            <id>xjc</id>
            <goals>
                <goal>xjc</goal>
            </goals>
        </execution>
    </executions>
    <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

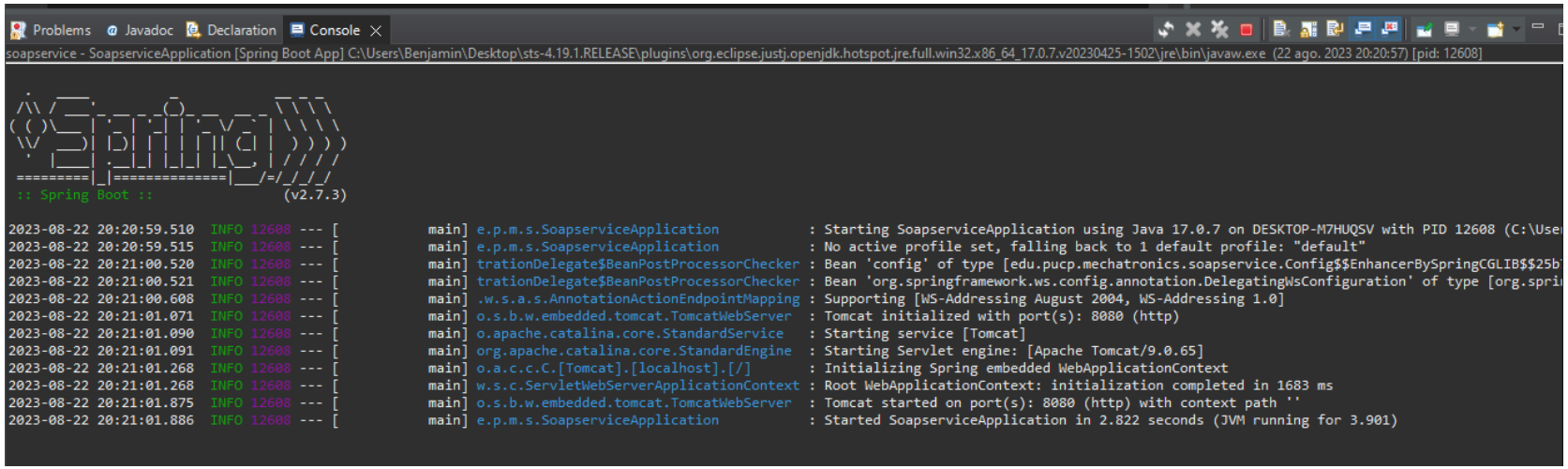
```
package edu.pucp.mechatronics.soapservice;

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);
    }
}
```

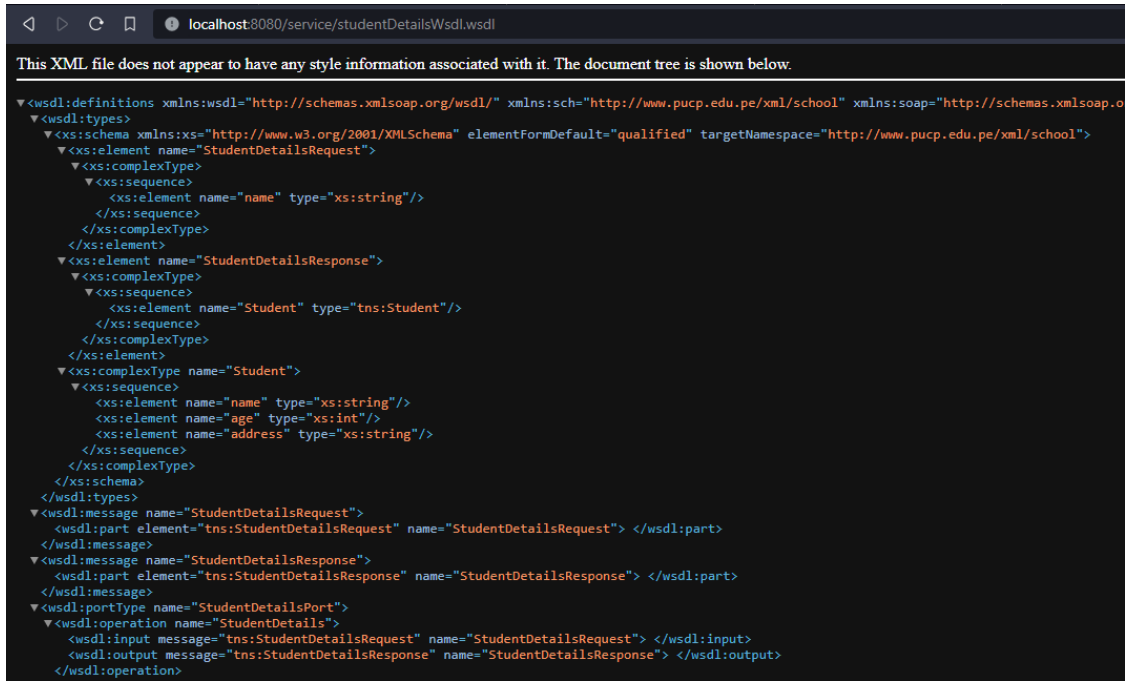
Corriendo el Servidor (1/2)

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



Corriendo el Servidor (2/2)

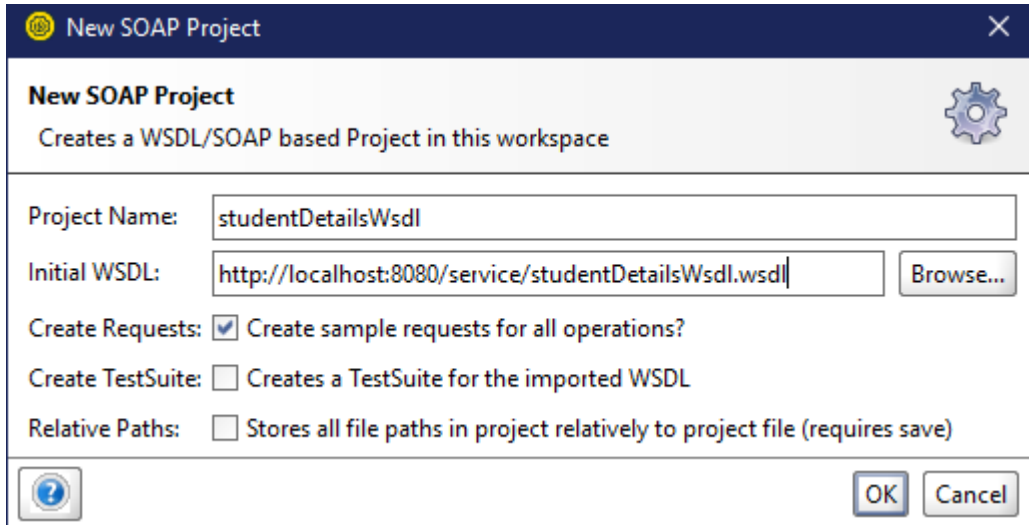
Se puede ver que el servidor esta corriendo correctamente en este link : http://localhost:8080/service/studentDetailsWsdl.wsdl
Aquí se podrá ver el contenido del archivo .wsdl



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.

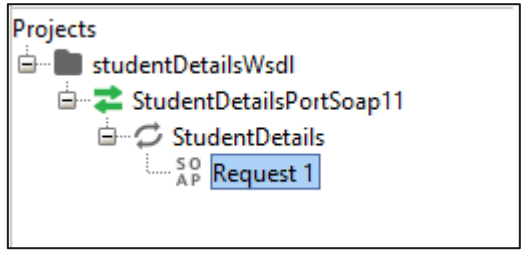


Probando el proyecto (2/2)

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/en
   <soapenv:Header/>
   <soapenv:Body>
      <sch:StudentDetailsRequest>
         <sch:name>Felix</sch:name>
      </sch:StudentDetailsRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

XML Raw

```
<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

http://stored.com

Customers — http://stored.com/api/customers

Products — http://stored.com/api/products

Sales — http://stored.com/api/sales

resource

Discounts — http://stored.com/api/discounts

http://api.stored.com

Metodos a utilizar

Get Customers

Request GET /api/customers

Response

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

Metodos a utilizar

Metodos a utilizar

Create a Customer

Request POST /api/customers

{ name: 'Kim'}

Response

{ id: 1, name: 'Kim'}

Metodos a utilizar

Update a Customer

Request PUT /api/customers/1

{ name: 'Fernando'}

Response

{ id: 1, name: 'Fernando'}

Metodos a utilizar

Delete a Customer

Request DELETE /api/customers/1

Response

{ id: 1, name: 'Fernando'}

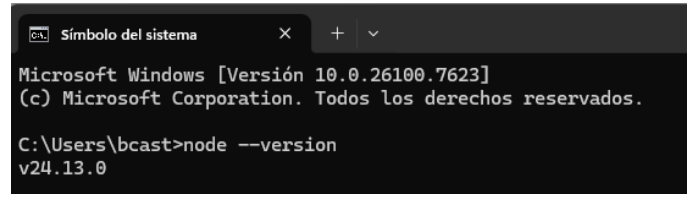
Requisitos

- 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

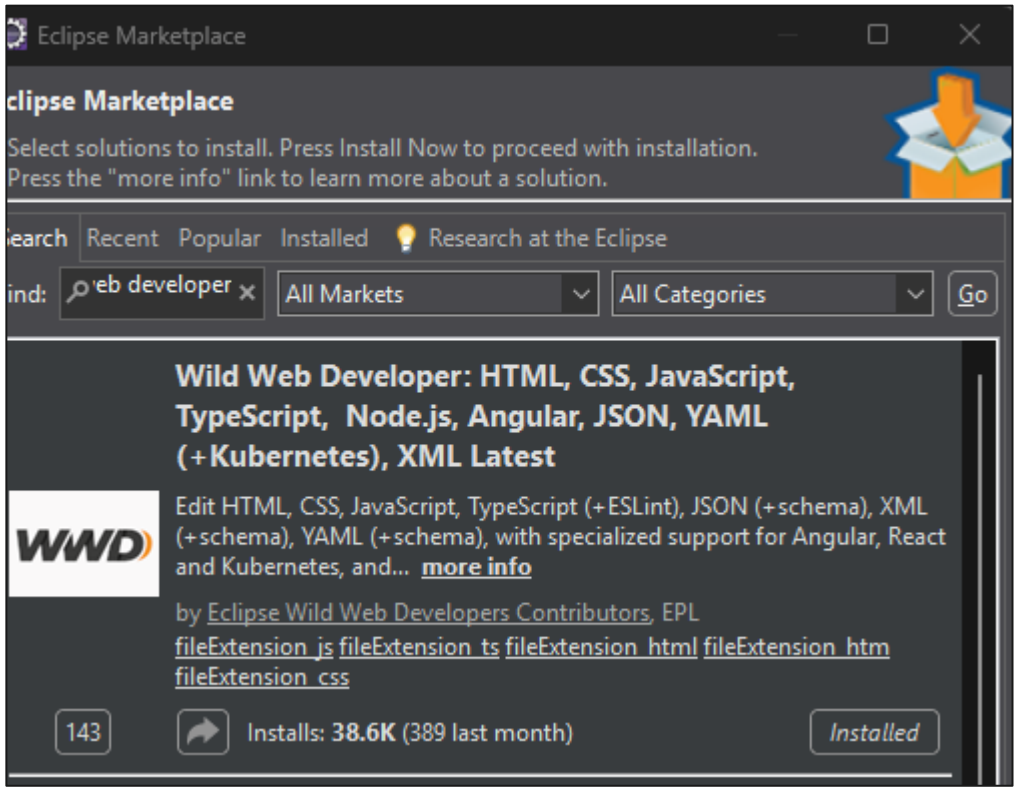


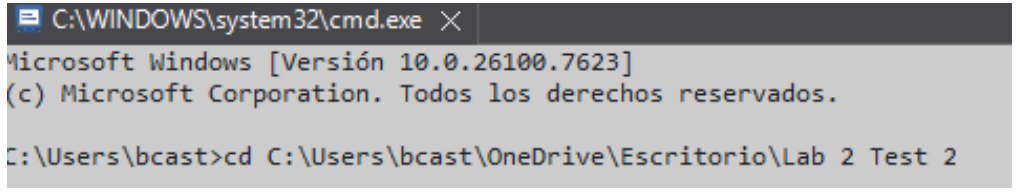
NPM

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



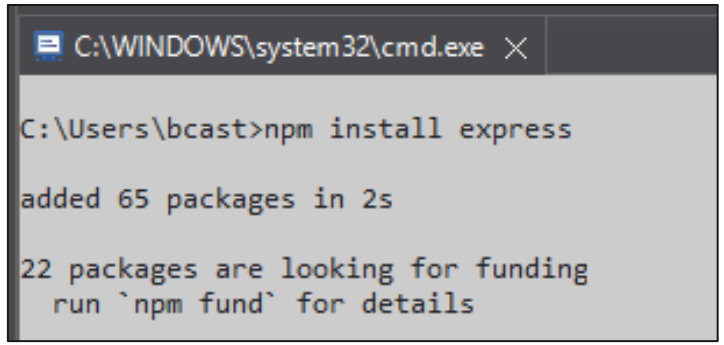


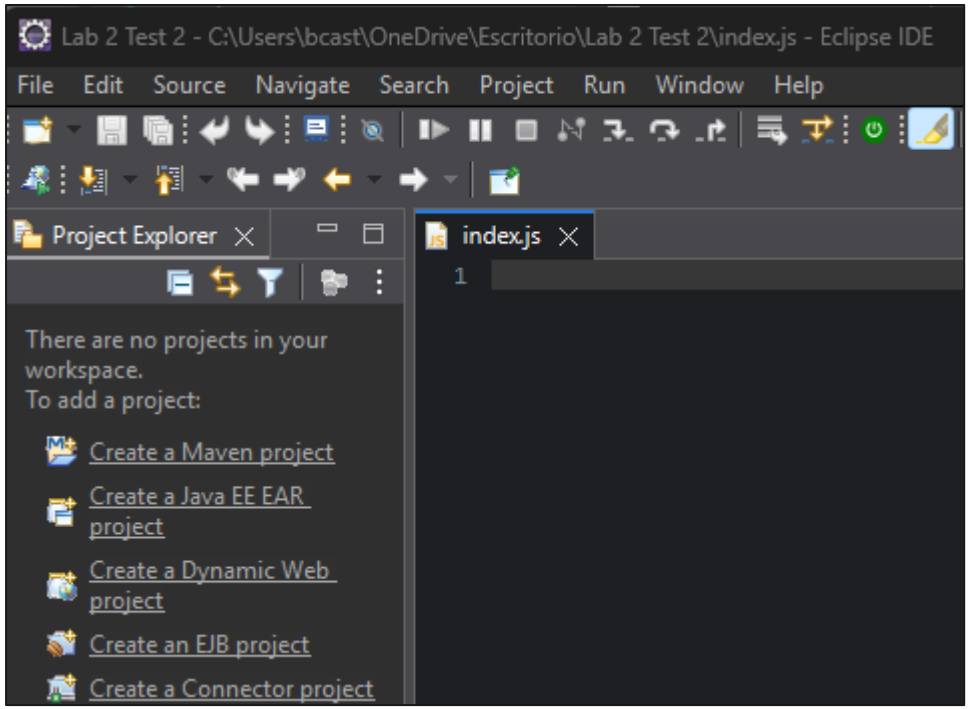
Configuración del proyecto

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

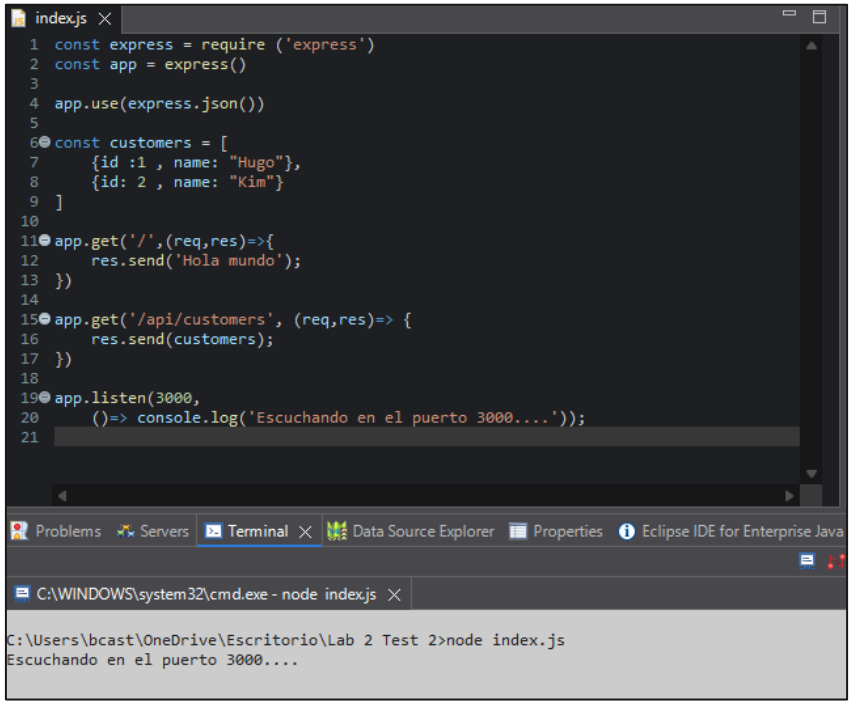
Para abrirlo simplemente nos vamos a File , Open File y lo buscamos.





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.



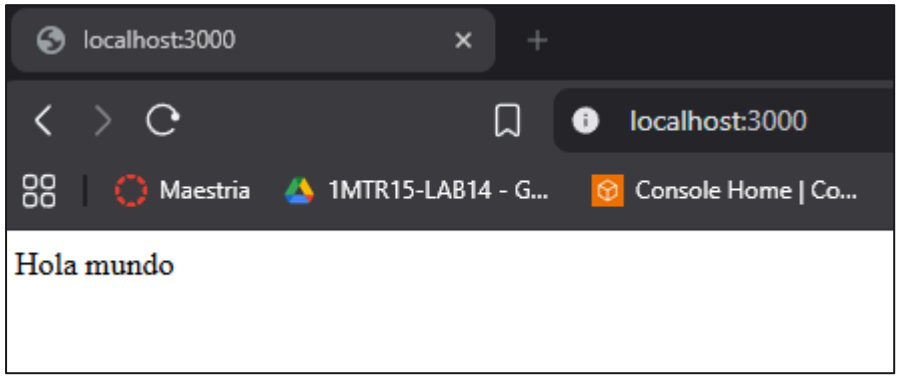

```
const express = require ('express')
const app = express()

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....'));
```

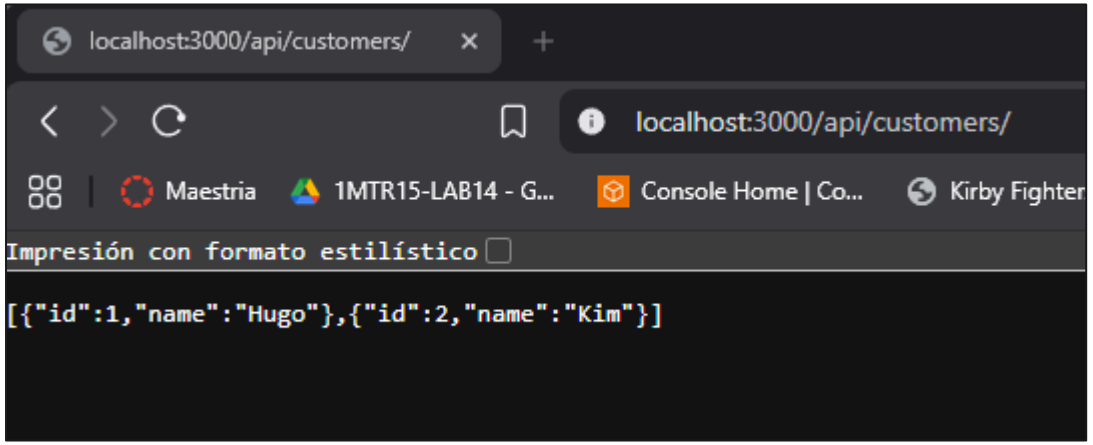
Configuración del proyecto

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

O

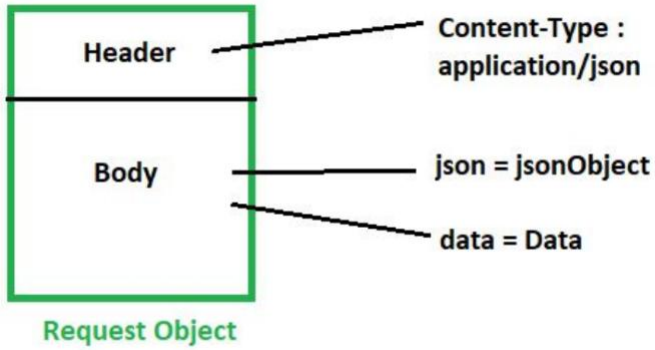
http://localhost:3000/api/customers/ para probar la información insertada.





Configuración del proyecto

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

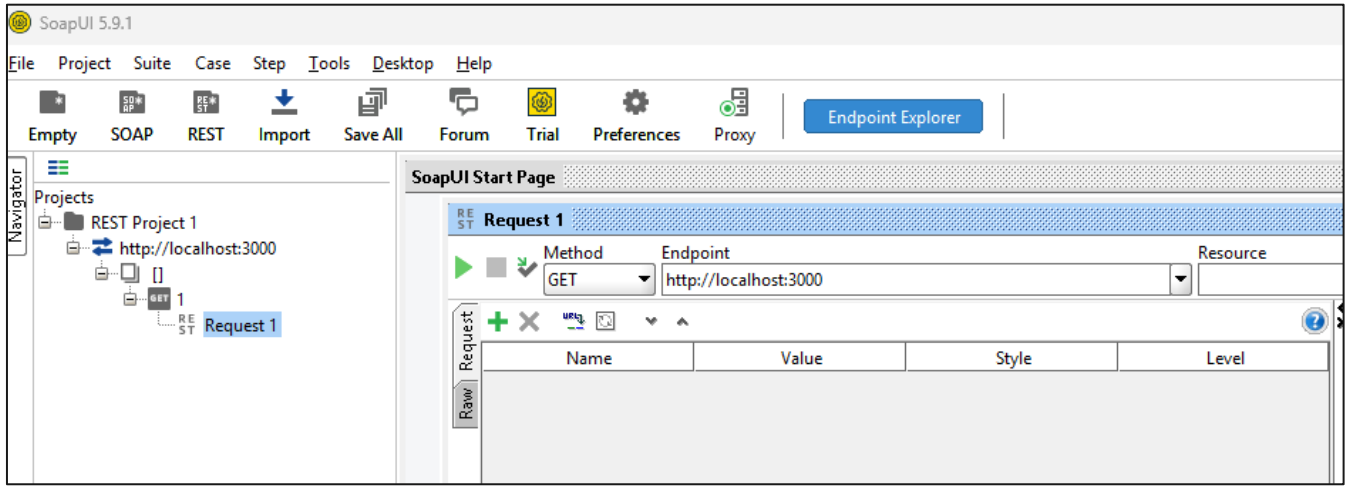



```
app.get('/api/customers/:id', (req,res)=> {
    const customer = customers.find(c => c.id ===parseInt(req.params.id));
    if(!customer){
        res.status(404).send('El usuario con este ID no se encuentra')
    }
    res.send(customer)
})

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



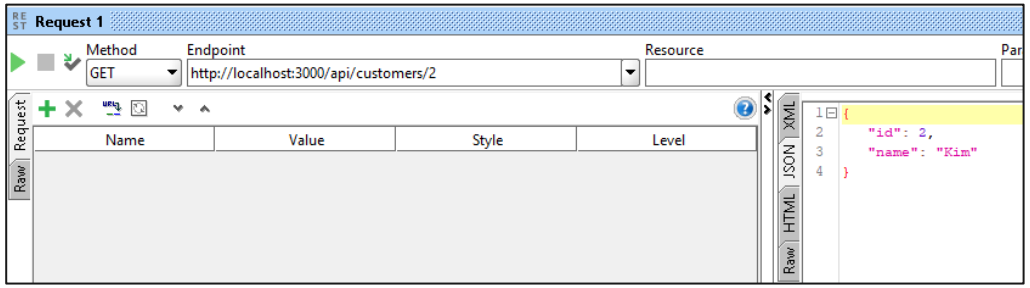
Probando el proyecto

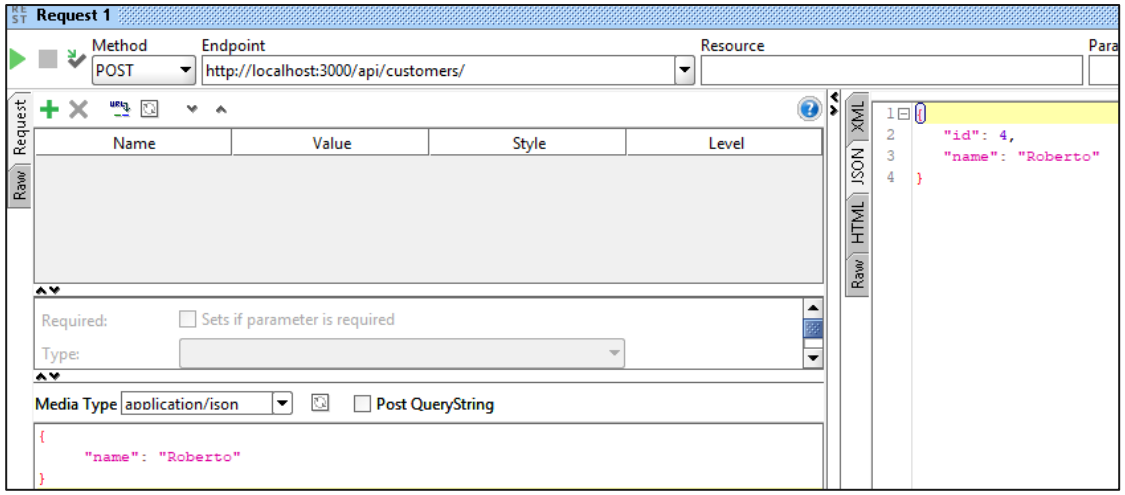
Una vez tengamos esto , haremos la prueba con los apis creados, para esto solo debemos seleccionar el método correspondiente e ir cambiando la URL y luego darle click a Play

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 correspodientes, en este caso el name

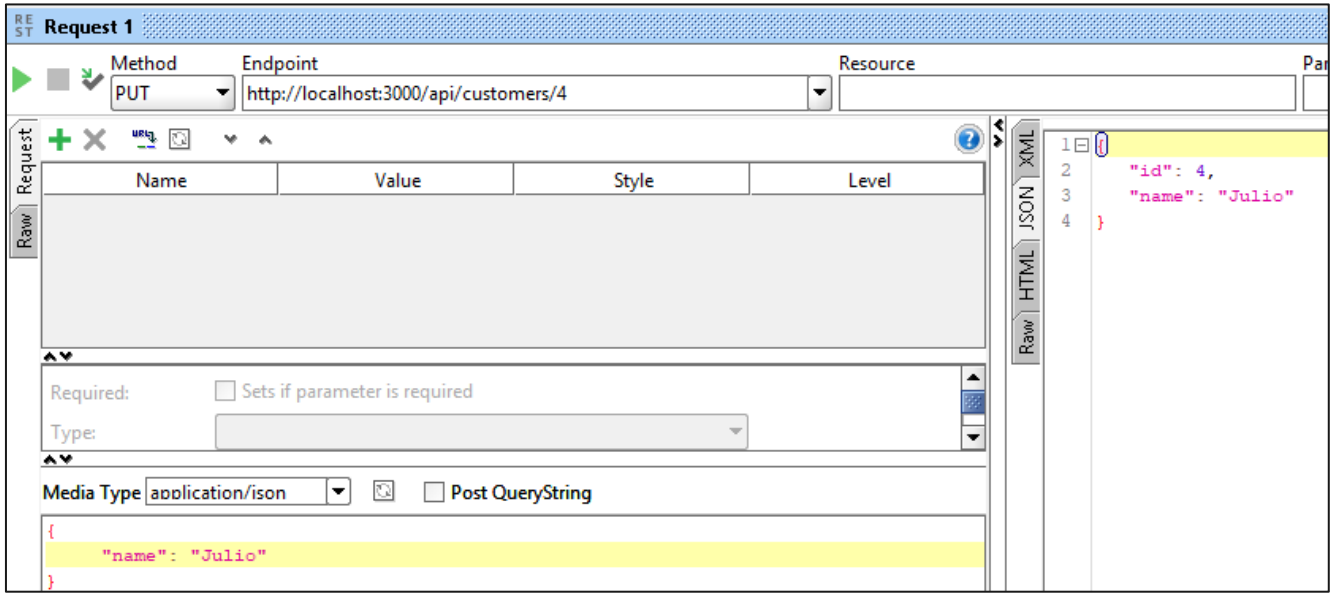




Probando el proyecto

Continuando, tambien implementaremos el método put

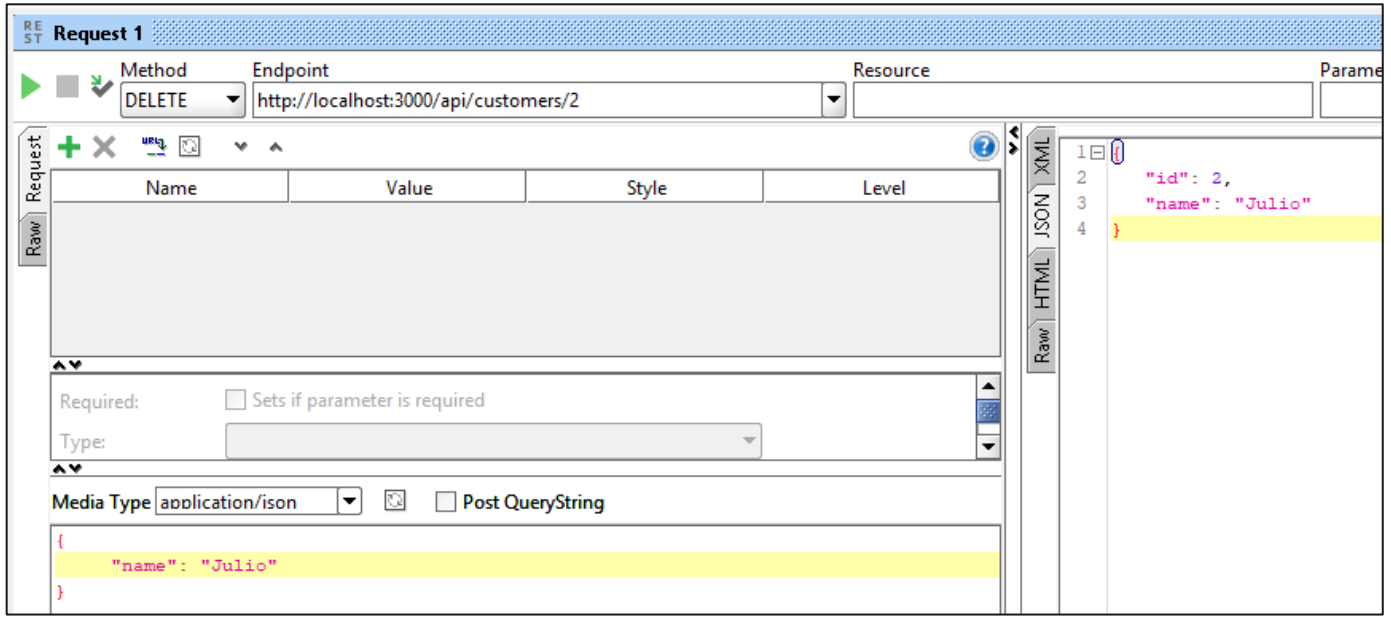
```
app.put('/api/customers/:id', (req,res)=> {
    const customer = customers.find(c => c.id
===parseInt(req.params.id));
    if(!customer){
        res.status(404).send('El usuario con este
ID no se encuentra')
    }
    if(!req.body.name ||
req.body.name.length<3){
        res.status(403).send("El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