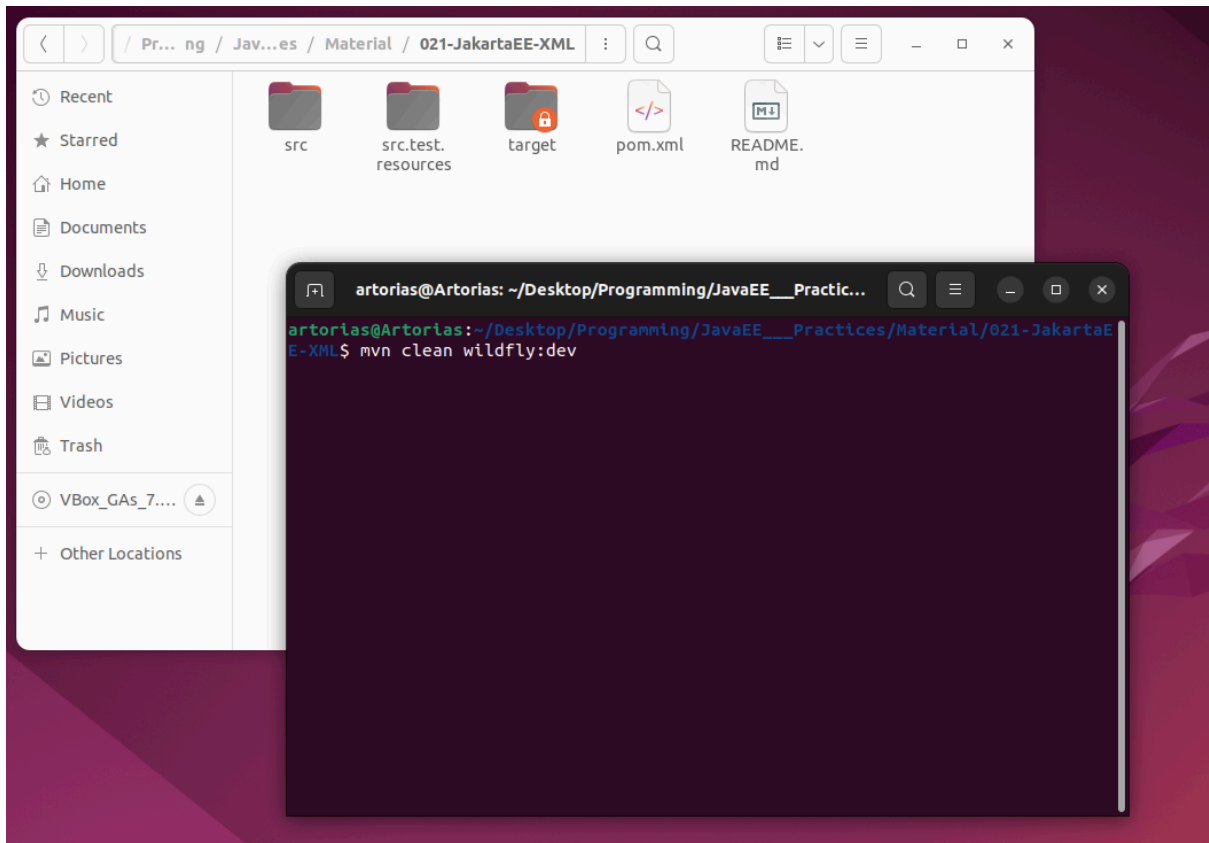


Ejercicio 3

WebServices

1) - En primer lugar, se deployea el codigo y se levanta el servidor. Para eso usamos el comando `mvn clean wildfly:dev`.



Una vez que el server esta levantado podemos ir a la URL que nos muestra en los logs del servidor , que es la URL de WSDL.

```
root@Artorias: /home/artorias/Desktop/Programming/JavaEE___Practices/Material/021-JakartaE...

2-api.jar in /home/artorias/Desktop/Programming/JavaEE___Practices/Material/021-JakartaEE-XML/target/021-JakartaEE-XML/WEB-INF/lib/ha-api-3.1.13.jar does not point to a valid jar for a Class-Path reference.
14:49:32,669 WARN [org.jboss.as.server.deployment] (MSC service thread 1-2) WFLYSRV0059: Class Path entry org.jboss.weld.servlet-api.jar in /home/artorias/Desktop/Programming/JavaEE___Practices/Material/021-JakartaEE-XML/target/021-JakartaEE-XML/WEB-INF/lib/ha-api-3.1.13.jar does not point to a valid jar for a Class-Path reference.
14:49:33,030 INFO [org.jboss.weld.deployer] (MSC service thread 1-5) WFLYWELD0003: Processing weld deployment 021-JakartaEE-XML.war
14:49:33,079 INFO [org.hibernate.validator.internal.util.Version] (MSC service thread 1-5) HV000001: Hibernate Validator 8.0.1.Final
14:49:33,335 INFO [org.jboss.weld.Version] (MSC service thread 1-4) WELD-000900: 5.1.2 (Final)
14:49:33,383 INFO [org.jboss.ws.cxf.metadata] (MSC service thread 1-4) JBWS024061: Adding service endpoint metadata: id=example00.interfaces.ws.soap.PagoSOAP
address=http://localhost:8080/021-JakartaEE-XML/PagoSOAP
implementor=example00.interfaces.ws.soap.PagoSOAP
serviceName={http://soap.ws.interfaces.example00/}PagoSOAPService
portName={http://soap.ws.interfaces.example00/}PagoSOAPPort
annotationWsdLocation=null
wsdlLocationOverride=null
mtomEnabled=false
14:49:33,688 INFO [org.apache.cxf.wsdl.service.factory.ReflectionServiceFactoryBean] (MSC service thread 1-4) Creating Service {http://soap.ws.interfaces.example00/}PagoSOAPService from class example00.interfaces.ws.soap.PagoSOAP
14:49:33,863 INFO [org.apache.cxf.endpoint.ServerImpl] (MSC service thread 1-4) Setting the server's published address to be http://localhost:8080/021-JakartaEE-XML/PagoSOAP
14:49:33,925 INFO [org.jboss.ws.cxf.deployment] (MSC service thread 1-4) JBWS024074: WSDL published to: file:/home/artorias/Desktop/Programming/JavaEE___Practices/Material/021-JakartaEE-XML/target/server/standalone/data/wsdl/021-JakartaEE-XML.war/PagoSOAPService.wsdl
14:49:33,964 INFO [org.jboss.as.webservices] (MSC service thread 1-4) WFLYWS0003: Starting service jboss.ws.endpoint."021-JakartaEE-XML.war"."example00.interfaces.ws.soap.PagoSOAP"
14:49:34,406 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 78) WFLYUT0021: Registered web context: '/021-JakartaEE-XML' for server 'default-server'
14:49:34,456 INFO [org.jboss.as.server] (management-handler-thread - 1) WFLYSRV0010: Deployed "021-JakartaEE-XML.war" (runtime-name : "021-JakartaEE-XML.war")
.S
```

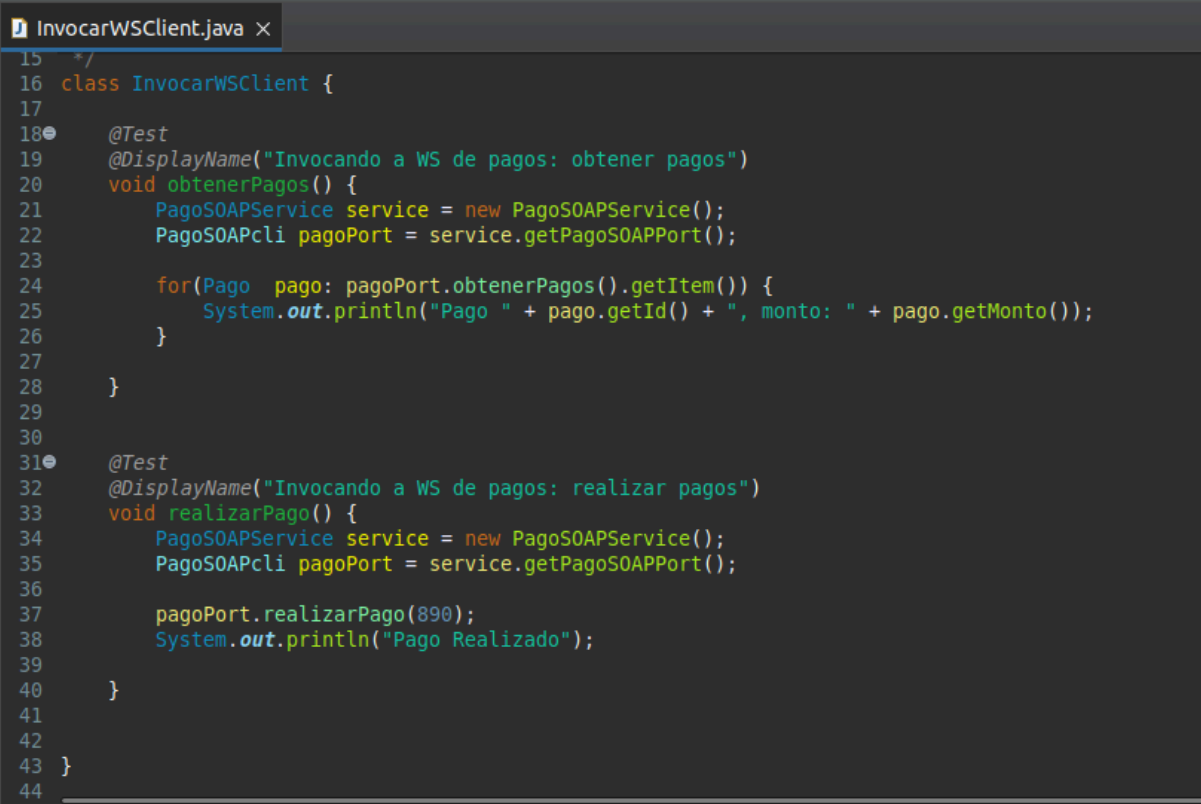
Ahora ponemos esa URL en el navegador agregandole ?wsdl y se vera lo siguiente.

De esta manera podemos ver correctamente el XML.

```
localhost:8080/021-JakartaEE-XML/PagoSOAP?wsdl

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'>
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://soap.ws.interfaces.example00/" xmlns:soap="http://schemas.xmlsoap.org/soap/http"
  xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="PagoSOAPService" targetNamespace="http://soap.ws.interfaces.example00/">
  <wsdl:types>
    <xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://soap.ws.interfaces.example00/" targetNamespace="http://soap.ws.interfaces.example00/" version="1.0">
      <xs:complexType name="Pago">
        <xs:sequence>
          <xs:element name="id" type="xs:int"/>
          <xs:element name="monto" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="PagoArray">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0" name="item" nillable="true" type="tns:Pago"/>
        </xs:sequence>
      </xs:complexType>
    </xsi:schema>
  </wsdl:types>
  <wsdl:message name="obtenerPagos"></wsdl:message>
  <wsdl:message name="realizarPagoResponse">
    <wsdl:part name="idPago" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="obtenerPagosResponse">
    <wsdl:part name="pagos" type="tns:PagoArray"/>
  </wsdl:message>
  <wsdl:message name="realizarPago">
    <wsdl:part name="monto" type="xsd:int"/>
  </wsdl:message>
  <wsdl:portType name="PagoSOAP">
    <wsdl:operation name="obtenerPagos">
      <wsdl:input message="tns:obtenerPagos" name="obtenerPagos"/>
      <wsdl:output message="tns:obtenerPagosResponse" name="obtenerPagosResponse"/>
    </wsdl:operation>
    <wsdl:operation name="realizarPago">
      <wsdl:input message="tns:realizarPago" name="realizarPago"/>
      <wsdl:output message="tns:realizarPagoResponse" name="realizarPagoResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="PagoSOAPServiceSoapBinding" type="tns:PagoSOAP">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="obtenerPagos">
      <soap:operation soapAction="" style="rpc"/>
      <wsdl:input name="obtenerPagos">
        <soap:body namespace="http://soap.ws.interfaces.example00/" use="literal"/>
      </wsdl:input>
      <wsdl:output name="obtenerPagosResponse">
        <soap:body namespace="http://soap.ws.interfaces.example00/" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="realizarPago">
      <soap:operation soapAction="" style="rpc"/>
      <wsdl:input name="realizarPago">
        <soap:body namespace="http://soap.ws.interfaces.example00/" use="literal"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
</definitions>
```

2) - Invocamos los test con el servidor corriendo en consola, desde eclipse.



```
15  */
16  class InvocarWSClient {
17
18      @Test
19      @DisplayName("Invocando a WS de pagos: obtener pagos")
20      void obtenerPagos() {
21          PagoSOAPService service = new PagoSOAPService();
22          PagoSOAPcli pagoPort = service.getPagoSOAPPort();
23
24          for(Pago pago: pagoPort.obtenerPagos().getItem()) {
25              System.out.println("Pago " + pago.getId() + ", monto: " + pago.getMonto());
26          }
27
28      }
29
30
31      @Test
32      @DisplayName("Invocando a WS de pagos: realizar pagos")
33      void realizarPago() {
34          PagoSOAPService service = new PagoSOAPService();
35          PagoSOAPcli pagoPort = service.getPagoSOAPPort();
36
37          pagoPort.realizarPago(890);
38          System.out.println("Pago Realizado");
39      }
40
41
42
43  }
44
```

Problems Servers Terminal Data Source Explorer Properties Console × JUnit PlantUML

<terminated> InvocarWSClient [JUnit] /home/lucasvm/Descargas/eclipse-jee-2024-03-R-linux-gtk-x86_64/eclipse/plugins/c
Pago Realizado
Pago 1, monto: 1000
Pago 2, monto: 2000
Pago 3, monto: 890

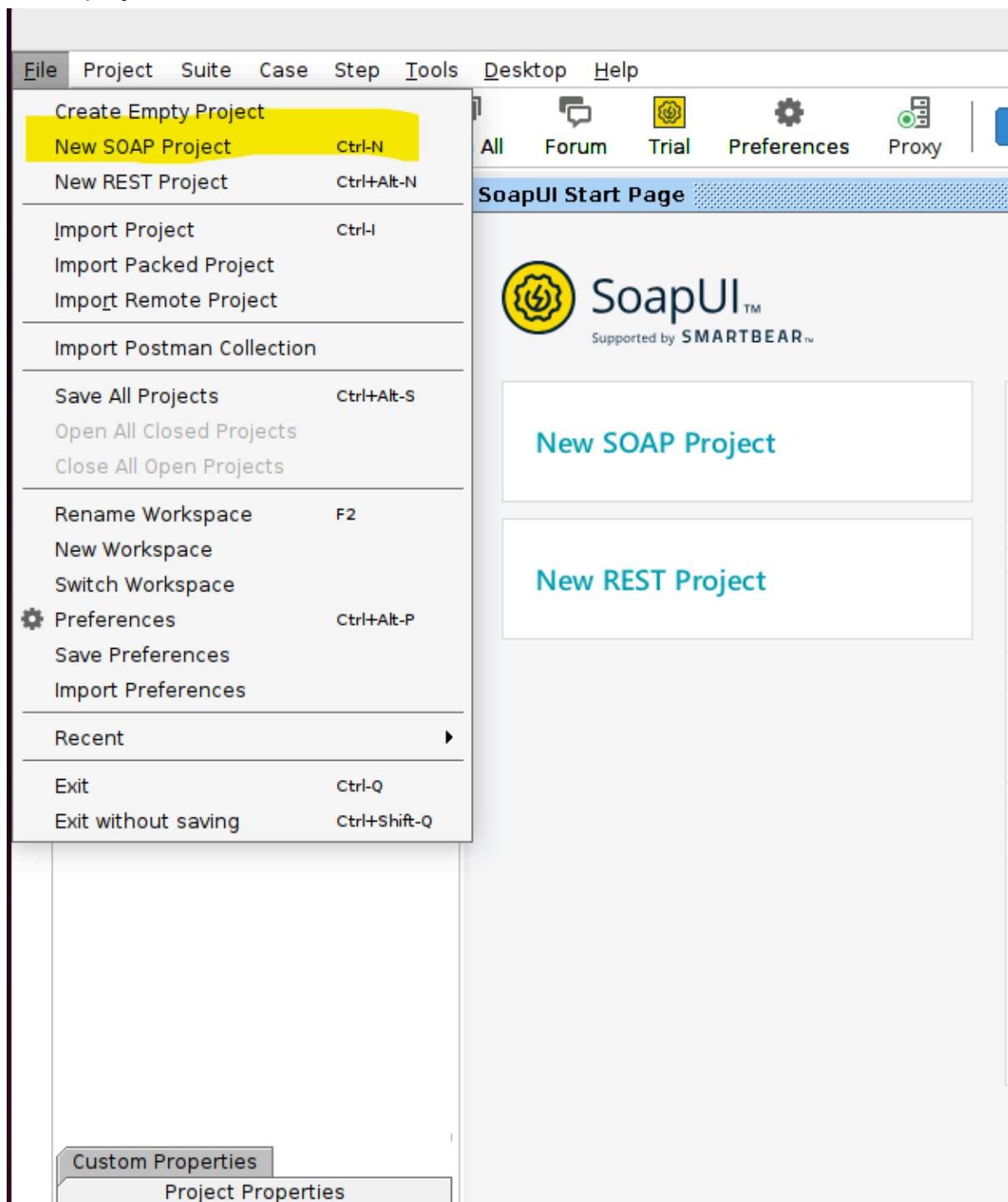
Los siguientes quedan de esta manera con el test. Funciona correctamente.

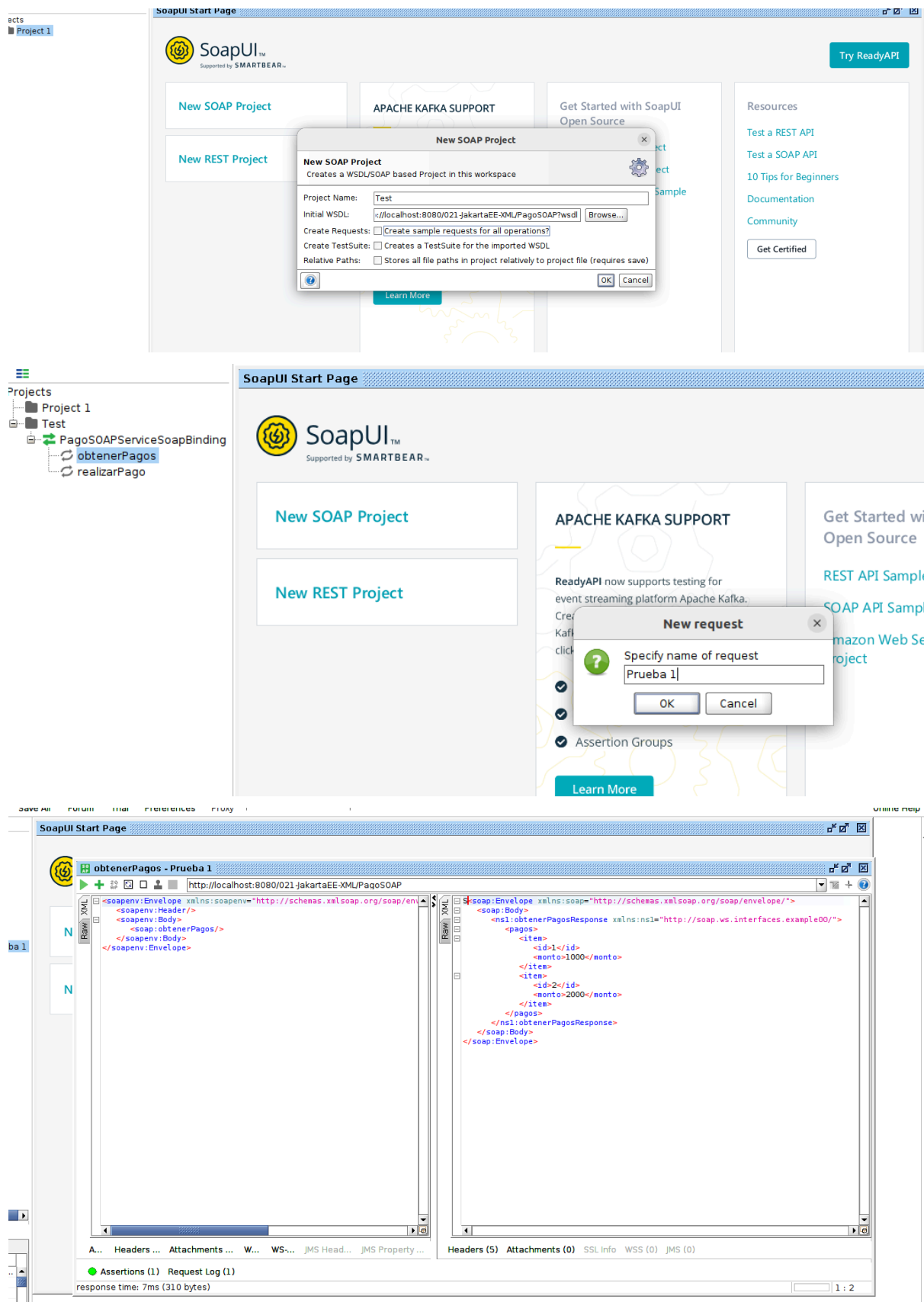
3) - Para el siguiente ejercicio, debemos instalar SOAP UI y ejecutar la bateria de test unitarios.

Quedará de la siguiente manera:

Con el servidor levantado en consola, abrimos SOAP UI y le damos a file, new

SOAP project.

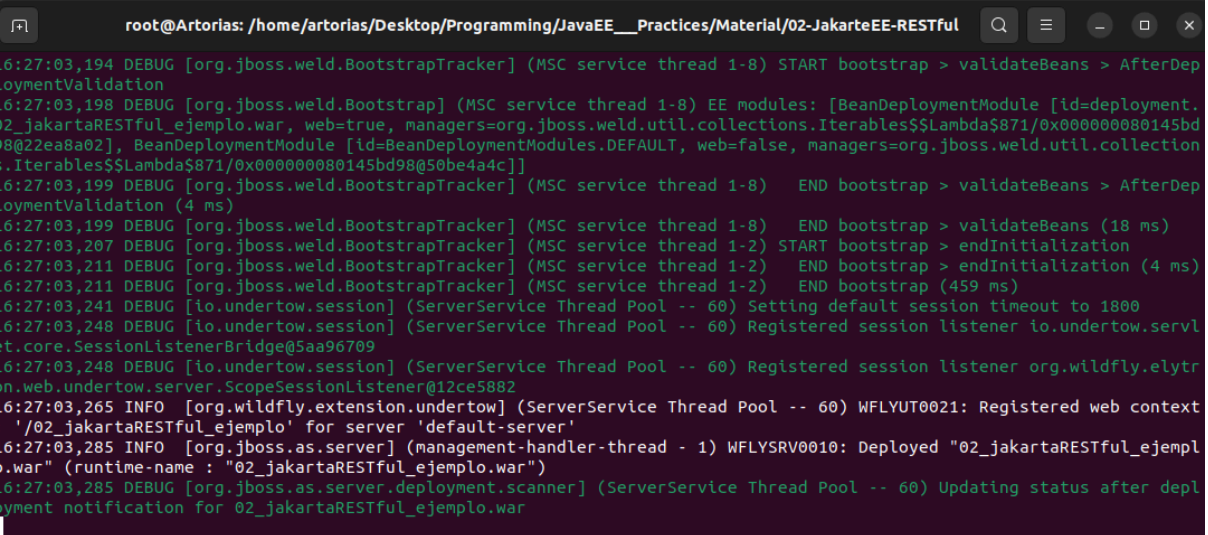




4) Y ahi vemos en el XML como los test funcionan correctamente. Hay un pago de 1000 y 2000.

RestFul

6) Para esta parte, una vez mas, vamos a los archivos proporcionados, donde encontraremos 02-JakartaRESTful. Realizamos los mismos pasos que anteriormente, con el mismo comando para levantar el servidor. Luego de usar mvn clean wildfly:dev queda de la siguiente manera, con el servidor deployado.



```
root@Artorias: /home/artorias/Desktop/Programming/JavaEE__Practices/Material/02-JakartaEE-RESTful
6:27:03,194 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-8) START bootstrap > validateBeans > AfterDeploymentValidation
6:27:03,198 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-8) EE modules: [BeanDeploymentModule [id=deployment.02_jakartaRESTful_ejemplo.war, web=true, managers=org.jboss.weld.util.collections.Iterables$$Lambda$871/0x000000080145bd8@22ea8a02], BeanDeploymentModule [id=BeanDeploymentModules.DEFAULT, web=false, managers=org.jboss.weld.util.collections.Iterables$$Lambda$871/0x000000080145bd98@50be4a4c]]
6:27:03,199 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-8) END bootstrap > validateBeans > AfterDeploymentValidation (4 ms)
6:27:03,199 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-8) END bootstrap > validateBeans (18 ms)
6:27:03,207 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-2) START bootstrap > endInitialization
6:27:03,211 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-2) END bootstrap > endInitialization (4 ms)
6:27:03,211 DEBUG [org.jboss.weld.BootstrapTracker] (MSC service thread 1-2) END bootstrap (459 ms)
6:27:03,241 DEBUG [io.undertow.session] (ServerService Thread Pool -- 60) Setting default session timeout to 1800
6:27:03,248 DEBUG [io.undertow.session] (ServerService Thread Pool -- 60) Registered session listener io.undertow.servlet.core.SessionListenerBridge@5aa96709
6:27:03,248 DEBUG [io.undertow.session] (ServerService Thread Pool -- 60) Registered session listener org.wildfly.elytron.web.undertow.server.ScopeSessionListener@12ce5882
6:27:03,265 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 60) WFLYUT0021: Registered web context '/02_jakartaRESTful_ejemplo' for server 'default-server'
6:27:03,285 INFO [org.jboss.as.server] (management-handler-thread - 1) WFLYSRV0010: Deployed "02_jakartaRESTful_ejemplo.war" (runtime-name : "02_jakartaRESTful_ejemplo.war")
6:27:03,285 DEBUG [org.jboss.as.server.deployment.scanner] (ServerService Thread Pool -- 60) Updating status after deployment notification for 02_jakartaRESTful_ejemplo.war
```

7) Una vez que se ejecutan los test quedan de la siguiente manera:

```
ConsumidorAPI_01_JakartaClientTest.java x
36     .target("http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes")
37     .request(MediaType.APPLICATION_JSON)
38     .get(String.class);
39
40     System.out.println(respuesta);
41
42 }
43
44 @Test
45 @DisplayName("Pasando parámetros http, invocando API con estilo rpc ")
46 void pasajeParametrosHttp() {
47
48     Client cliente = ClientBuilder.newClient();
49
50     String respuesta = cliente
51         .target("http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes")
52         .path("getClientes")
53         .queryParams("id", "1")
54         .request(MediaType.APPLICATION_JSON)
55         .get(String.class);
56
57     System.out.println(respuesta);
58
59 }
60
61
62 @Test
63 @DisplayName("Parseando respuesta ")
64 void parseoDeRespuesta() {
65
```

Problems Servers Terminal Data Source Explorer Properties Console x JUnit Plan

```
<terminated> ConsumidorAPI_01_JakartaClientTest [JUnit] /home/lucasvm/Descargas/eclipse-jee-2024-03-R-linux
Cliente [id=2, nombre=Mirta]
{"id":1,"nombre":"Luis"}
[{"id":1,"nombre":"Luis"}, {"id":2,"nombre":"Mirta"}]
```

8) Luego de instalar Postman, debemos colocar la URL para acceder al recurso RESTful, y sera de la manera de localhost - puerto - ruta de recurso - api/ - endpoint. Cabe mencionar que utilizaremos en metodo GET para obtener los recursos del servidor que tenemos corriendo. Quedando de la siguiente manera

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes`. The response is a JSON array with two objects, each containing an `id` and a `nombre`.

Query Params

Key	Value	Bulk Edit
Key	Value	

Body | Cookies | Headers (4) | Test Results

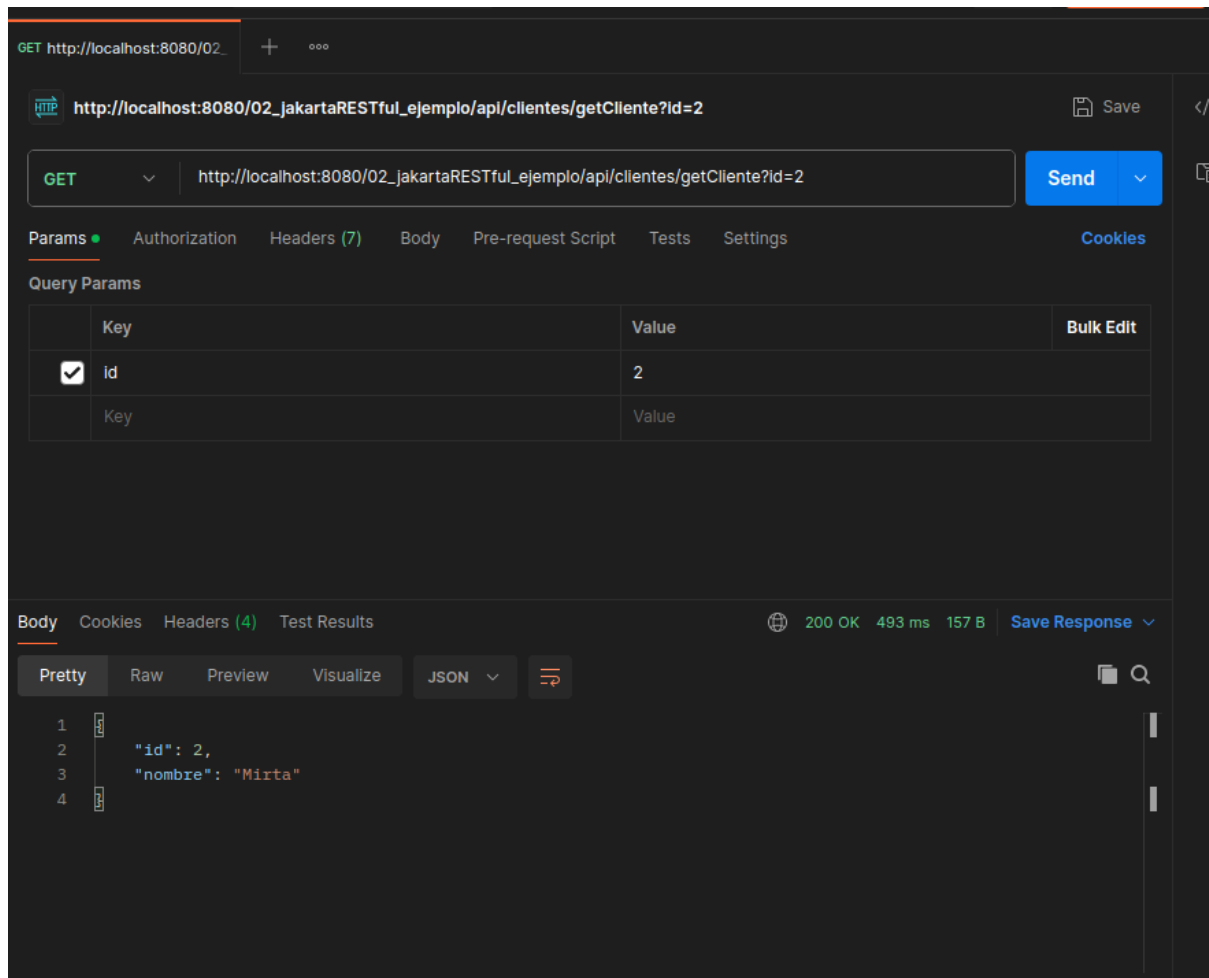
200 OK 9 ms 184 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2   {
3     "id": 1,
4     "nombre": "Luis"
5   },
6   {
7     "id": 2,
8     "nombre": "Mirta"
9   }
10 }
```

Como podemos ver, nos recupero 2 elementos , que los devuelve como un JSON, en pares de key : values.

9) De manera RPC:



10) Primero que nada, en el proyecto API RESTful necesitamos ir hasta la implementacion en `ClientServiceImpl` y desde ahi, podemos implementar el metodo correspondiente. Quedando de la siguiente manera:

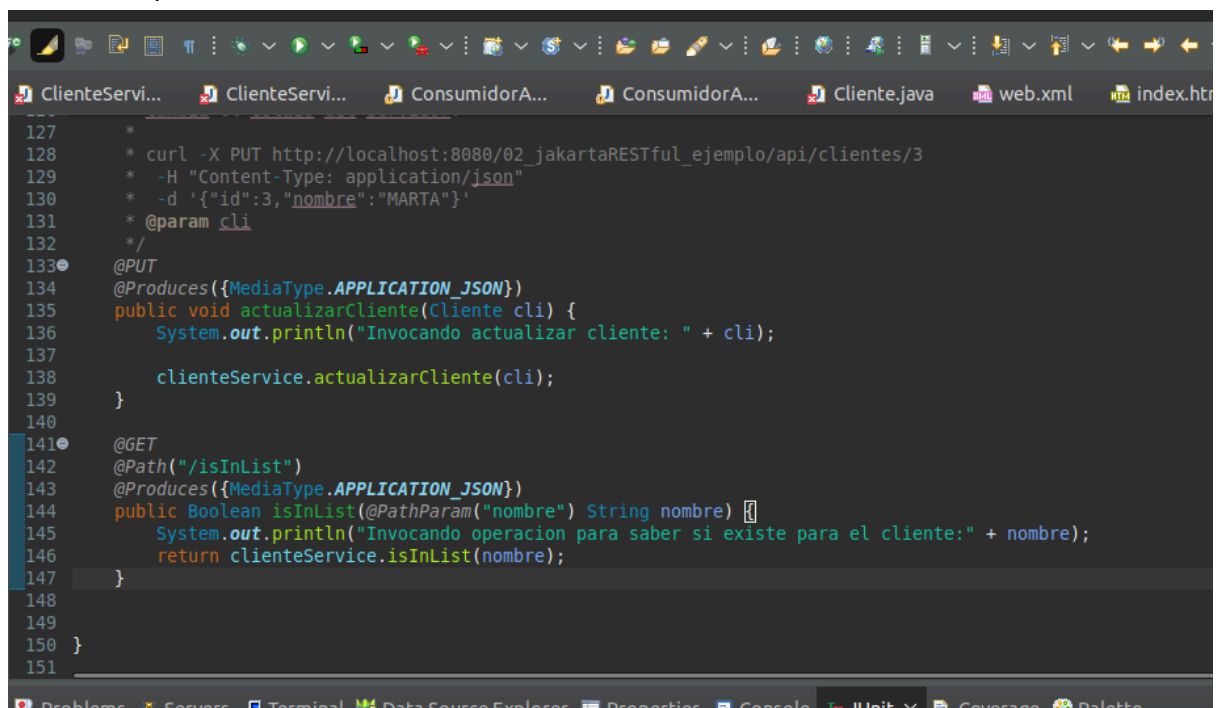


y no podemos olvidar agregarlo a los metodos.



```
1 package ejemplo00.aplicacion;
2
3 import java.util.List;
4
5
6
7 public interface ClienteService {
8     public List<Cliente> obtenerClientes();
9     public Cliente obtenerCliente(int id);
10    public void borrarCliente(int id);
11    public void actualizarCliente(Cliente cli);
12    public void insertar(Cliente cli);
13    public Boolean isInList(String nombre);
14 }
```

Tambien implementamos un método de manera RESTful:

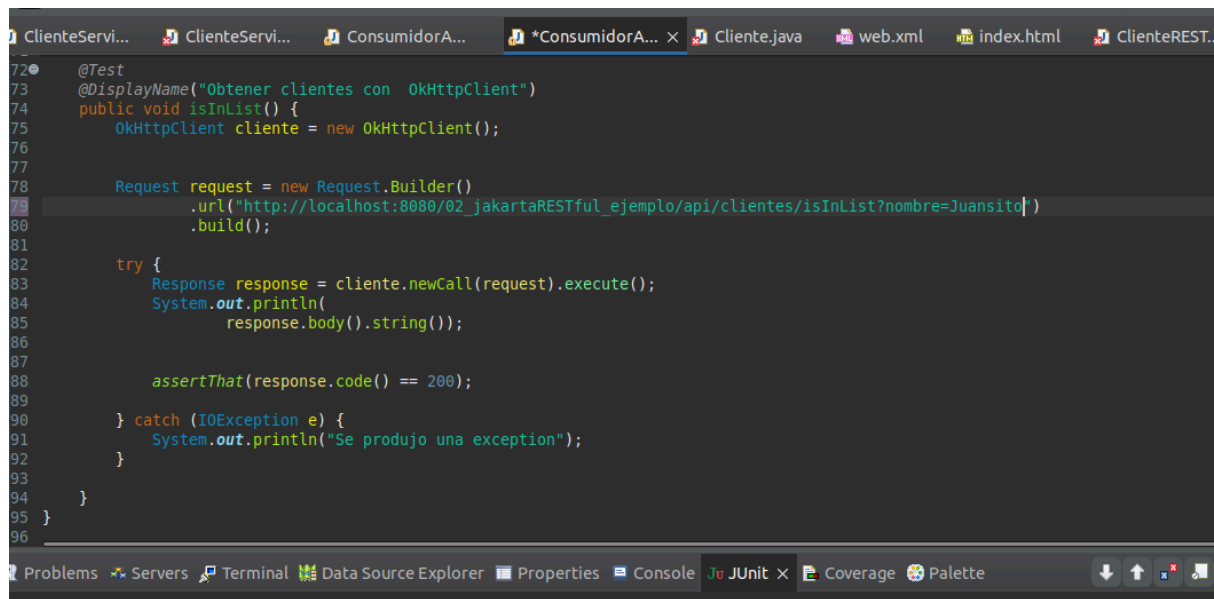


```
127 *
128 * curl -X PUT http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes/3
129 * -H "Content-Type: application/json"
130 * -d '{"id":3,"nombre":"MARTA"}'
131 * @param cli
132 */
133 @PUT
134 @Produces({MediaType.APPLICATION_JSON})
135 public void actualizarCliente(Cliente cli) {
136     System.out.println("Invocando actualizar cliente: " + cli);
137
138     clienteService.actualizarCliente(cli);
139 }
140
141 @GET
142 @Path("/isInList")
143 @Produces({MediaType.APPLICATION_JSON})
144 public Boolean isInList(@PathParam("nombre") String nombre) {
145     System.out.println("Invocando operacion para saber si existe para el cliente:" + nombre);
146     return clienteService.isInList(nombre);
147 }
148
149
150 }
151
```

11) Seguido este caso debemos implementar el test unitario.

Podemos tomar algunos de los ejemplos anteriores y simplemente adecuarlo a nuestra conveniencia.

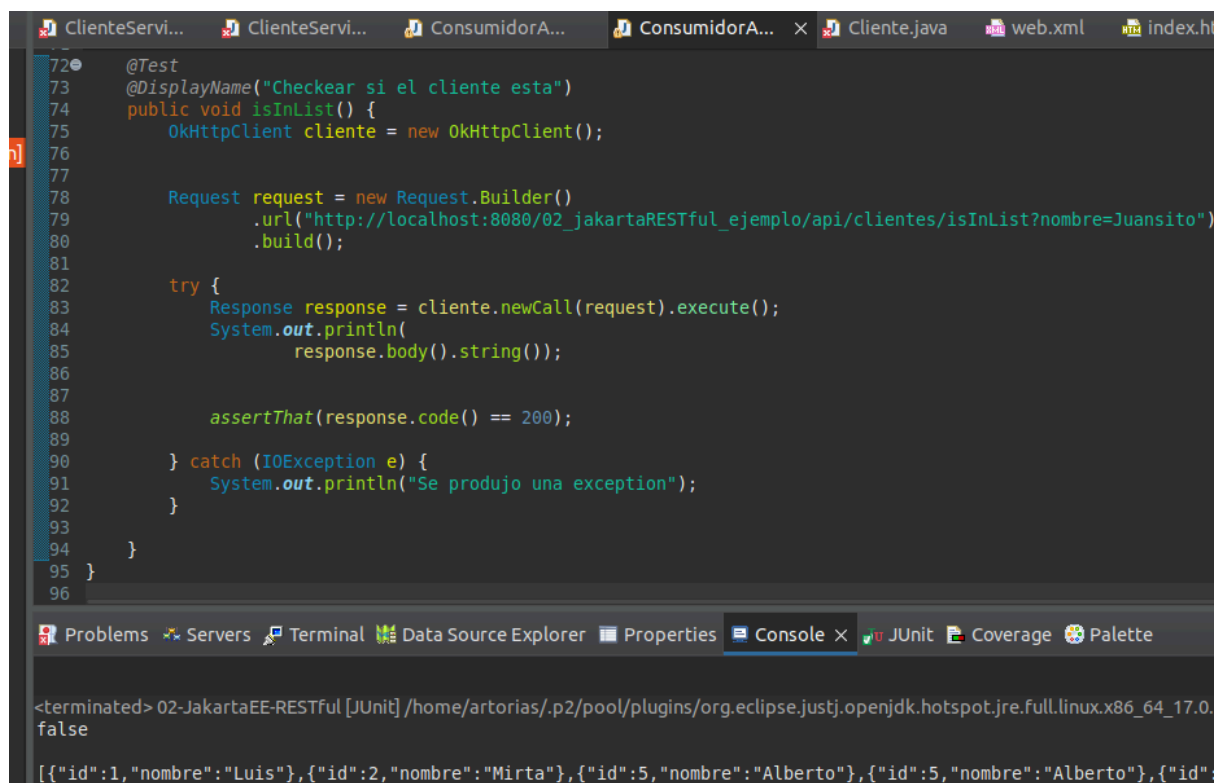
Quedando de la siguiente manera:



```
72 @Test
73 @DisplayName("Obtener clientes con OkHttpClient")
74 public void isInList() {
75     OkHttpClient cliente = new OkHttpClient();
76
77     Request request = new Request.Builder()
78         .url("http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes/isInList?nombre=Juansito")
79         .build();
80
81     try {
82         Response response = cliente.newCall(request).execute();
83         System.out.println(
84             response.body().string());
85
86         assertThat(response.code() == 200);
87
88     } catch (IOException e) {
89         System.out.println("Se produjo una exception");
90     }
91 }
92
93
94
95
96
```

Seguido de esto ejecutamos los test:

Vemos que para el test corriendo para buscar el user “Juansito”, nos arroja false, porque el usuario no se encuentra en la lista de usuarios.

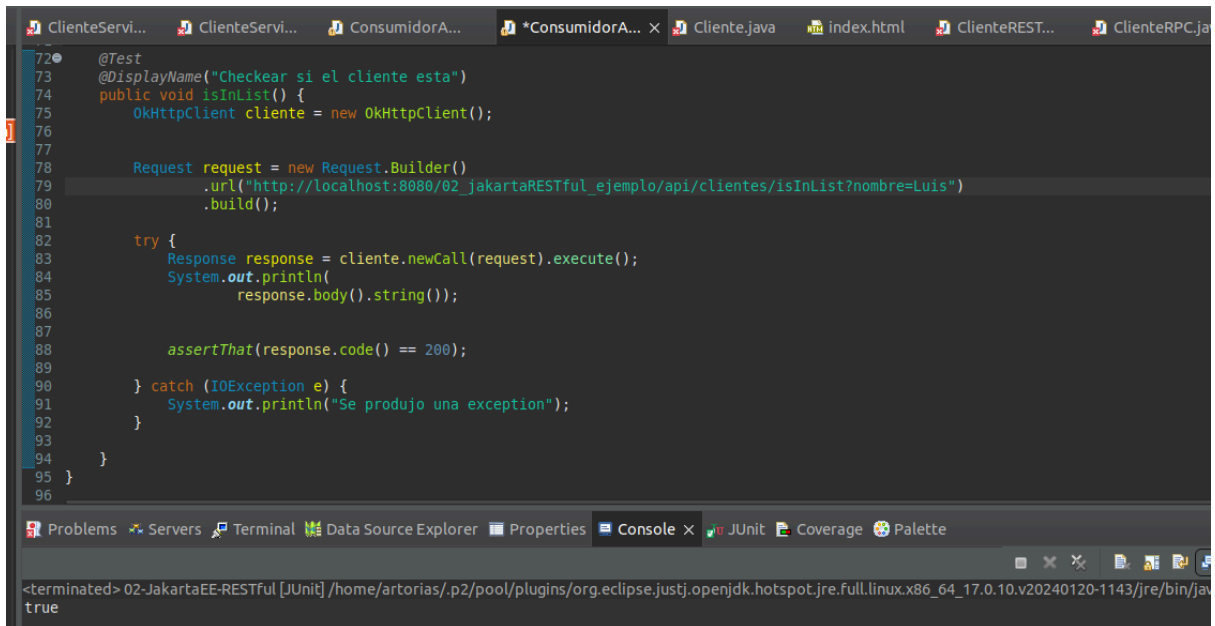


```
72 @Test
73 @DisplayName("Checkear si el cliente esta")
74 public void isInList() {
75     OkHttpClient cliente = new OkHttpClient();
76
77     Request request = new Request.Builder()
78         .url("http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes/isInList?nombre=Juansito")
79         .build();
80
81     try {
82         Response response = cliente.newCall(request).execute();
83         System.out.println(
84             response.body().string());
85
86         assertThat(response.code() == 200);
87
88     } catch (IOException e) {
89         System.out.println("Se produjo una exception");
90     }
91 }
92
93
94
95
96
```

<terminated> 02-JakartaEE-RESTful [JUnit] /home/artorias/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.
false

[{"id":1,"nombre":"Luis"}, {"id":2,"nombre":"Mirta"}, {"id":5,"nombre":"Alberto"}, {"id":5,"nombre":"Alberto"}, {"id":

Lo mismo podemos ejecutar para un user que si este en al lista de users.
Ejemplo, en este caso ejecutemos con “Luis”.



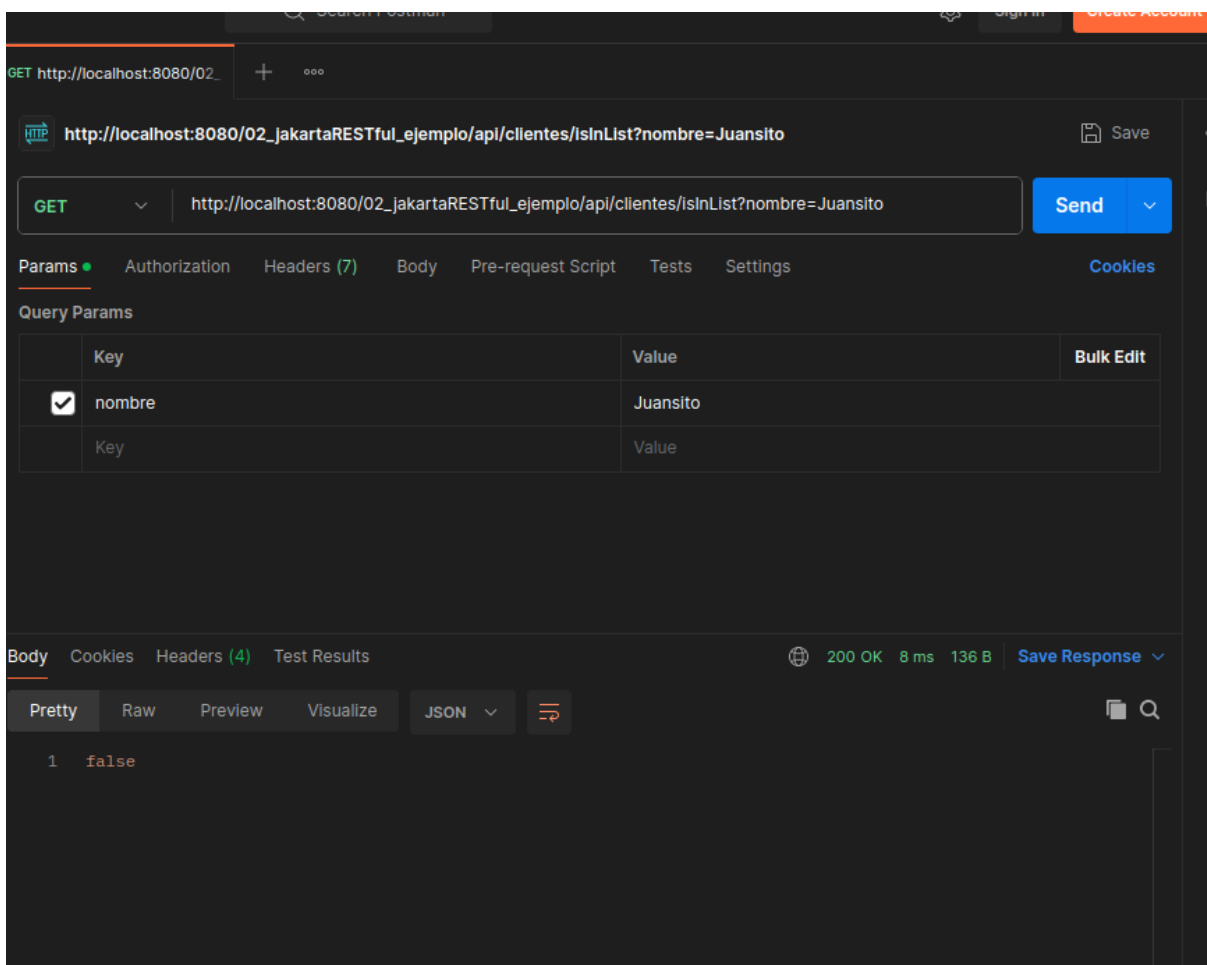
```
72 @Test
73 @DisplayName("Checkear si el cliente esta")
74 public void isInList() {
75     OkHttpClient cliente = new OkHttpClient();
76
77
78     Request request = new Request.Builder()
79         .url("http://localhost:8080/02_jakartaRESTful_ejemplo/api/clientes/isInList?nombre=Luis")
80         .build();
81
82     try {
83         Response response = cliente.newCall(request).execute();
84         System.out.println(
85             response.body().string());
86
87         assertThat(response.code() == 200);
88     } catch (IOException e) {
89         System.out.println("Se produjo una exception");
90     }
91 }
92
93
94
95 }
```

Problems Servers Terminal Data Source Explorer Properties Console x JUnit Coverage Palette

<terminated> 02-JakartaEE-RESTful [JUnit] /home/artorias/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.10.v20240120-1143/jre/bin/jav
true

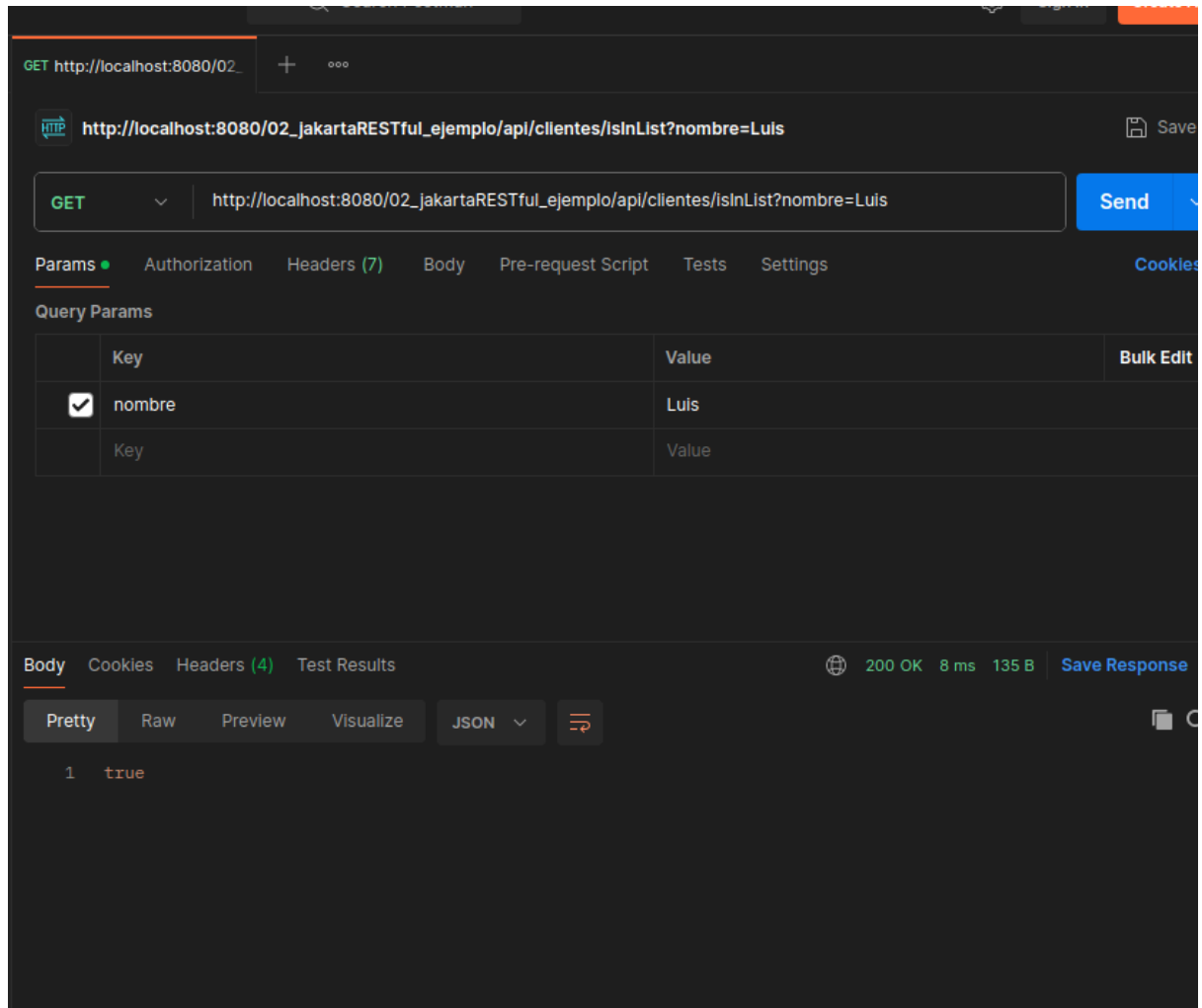
En este caso vemos que es True.

12) Haremos lo mismo con Postman, donde para el primer caso , hacemos la llamada para el user “Juansito”.



Y ahi vemos como retorna false.

Pero si llamamos a “Luis”, debería retornados true.



Como vemos, funciona correctamente.

Swagger

- 1) Primero bajamos el proyecto proporcionado por el docente.
- 2) Primero deployamos el servidor de la misma manera que las veces pasadas con `mvn clean wildfly:dev`.

Tenemos el servidor levantado.

```
root@Artorias: /home/artorias/Desktop/Programming/JavaEE__Practices/Material/02-JakartaEE-RESTful-Swagger
1) started in 1975ms - Started 196 of 288 services (132 services are lazy, passive or on-demand) - Server configuration file
in use: standalone.xml
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/artorias/Desktop/Programming/JavaEE__Practices/Material/02-JakartaEE-REST
ful-Swagger/src/main/resources
[INFO] Changes detected - recompiling the module! :dependency
[INFO] Compiling 8 source files with javac [debug release 17] to target/classes
[INFO] Exploding webapp
[INFO] Assembling webapp [02-JakartaEE-RESTful-Swagger] in [/home/artorias/Desktop/Programming/JavaEE__Practices/Material/
02-JakartaEE-RESTful-Swagger/target/02_jakartaRESTful_swagger]
[INFO] Processing war project
[INFO] Copying webapp resources [/home/artorias/Desktop/Programming/JavaEE__Practices/Material/02-JakartaEE-RESTful-Swagge
r/src/main/webapp]
17:31:00,220 INFO [org.jboss.as.server.deployment] (MSC service thread 1-3) WFLYSRV0027: Starting deployment of "02_jakarta
RESTful_swagger.war" (runtime-name: "02_jakartaRESTful_swagger.war")
17:31:01,049 INFO [org.jboss.weld.deployer] (MSC service thread 1-7) WFLYWELD0003: Processing weld deployment 02_jakartaRE
STful_swagger.war
17:31:01,223 INFO [org.hibernate.validator.internal.util.Version] (MSC service thread 1-7) HV000001: Hibernate Validator 8
.0.1.Final
17:31:01,397 WARN [org.jboss.as.jaxrs] (MSC service thread 1-8) WFLYRS0018: Explicit usage of Jackson annotation in a Jaka
rta RESTful Web Services deployment; the system will disable Jakarta JSON Binding processing for the current deployment. Co
nsider setting the 'resteasy.preferJacksonOverJsonB' property to 'false' to restore Jakarta JSON Binding.
17:31:01,417 INFO [org.jboss.weld.Version] (MSC service thread 1-8) WELD-000900: 5.1.2 (Final)
17:31:02,453 INFO [org.jboss.resteasy.resteasy_jaxrs.i18n] (ServerService Thread Pool -- 60) RESTEASY002225: Deploying jak
arta.ws.rs.core.Application: class api.MensajesApplicationConfig
17:31:02,456 WARN [org.jboss.as.weld] (ServerService Thread Pool -- 60) WFLYWELD0052: Using deployment classloader to load
proxy classes for module com.fasterxml.jackson.jakarta.jackson-jakarta-json-provider. Package-private access will not work
. To fix this the module should declare dependencies on [org.jboss.weld.core, org.jboss.weld.spi, org.jboss.weld.api]
17:31:02,547 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 60) WFLYUT0021: Registered web context: '
/02_jakartaRESTful_swagger' for server 'default-server'
17:31:02,595 INFO [org.jboss.as.server] (management-handler-thread - 1) WFLYSRV0010: Deployed "02_jakartaRESTful_swagger.w
ar" (runtime-name : "02_jakartaRESTful_swagger.war")
```

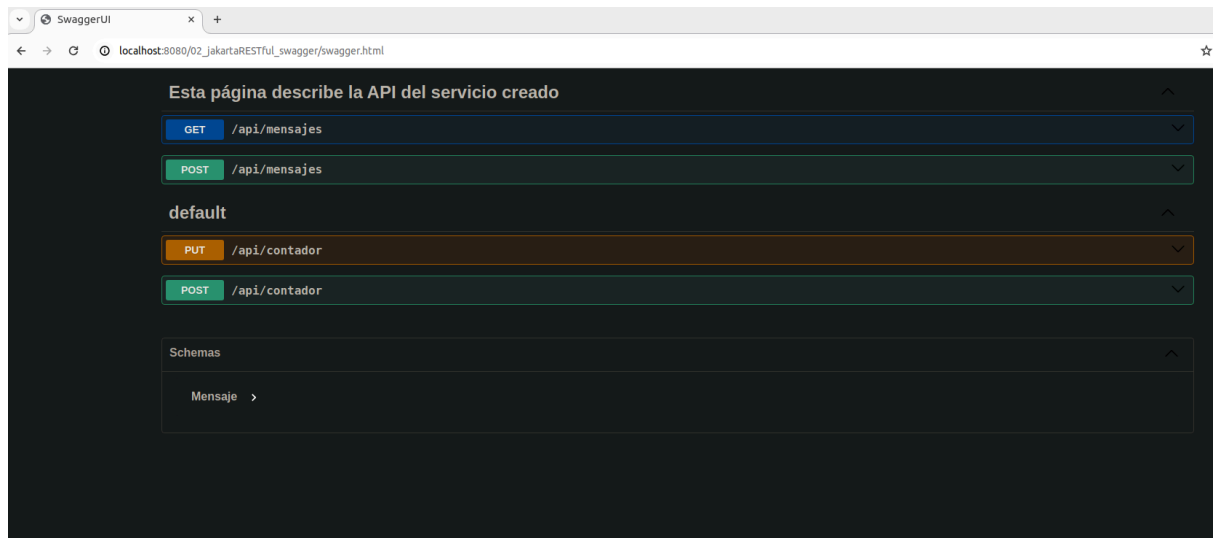
Luego podemos acceder al recurso.

http://localhost:8080/02_jakartaRESTful_swagger/api/openapi.json

Y nos mostrará los siguientes servicios deployados:

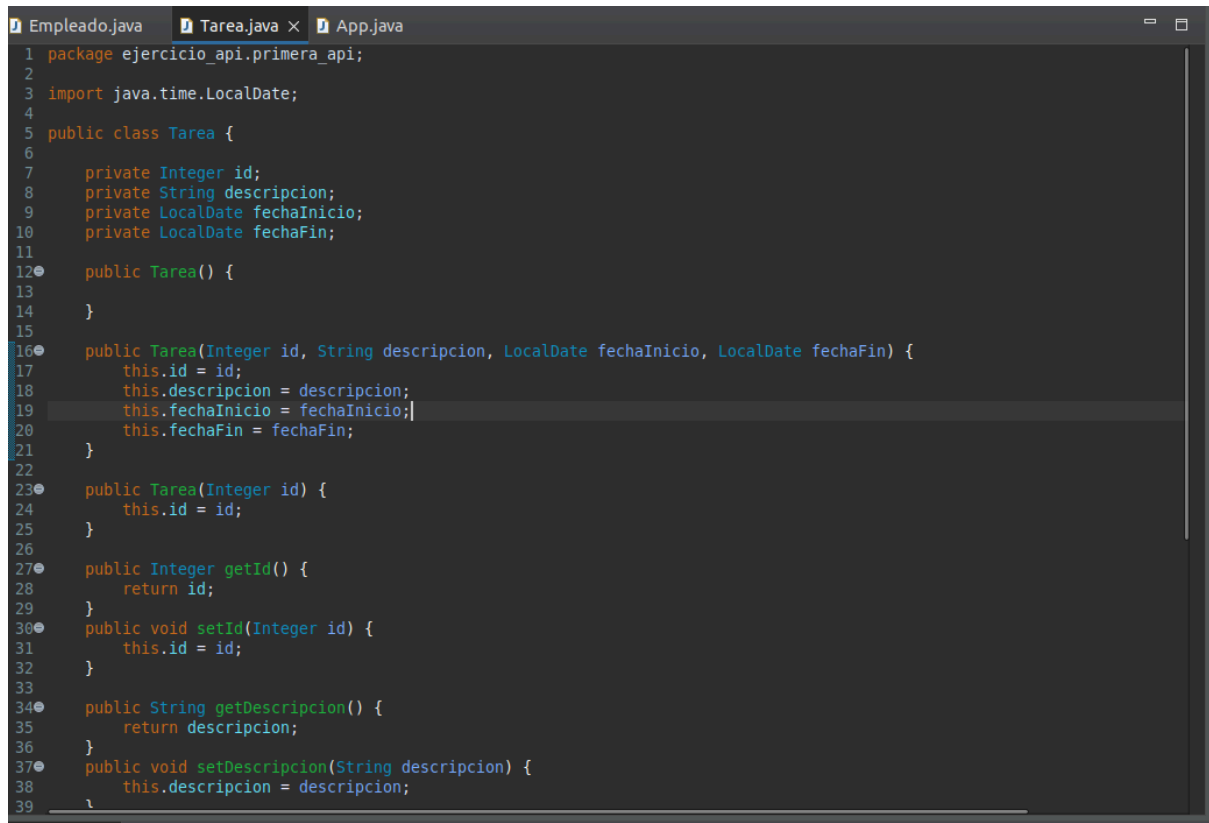
```
localhost:8080/02_jakartaRESTful_swagger/api/openapi.json
pretty print
{
  "openapi": "3.0.1",
  "paths": {
    "/api/mensajes": {
      "get": {
        "tags": [
          "Esta página describe la API del servicio creado"
        ],
        "operationId": "listar",
        "responses": {
          "default": {
            "description": "default response",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/Mensaje"
                  }
                }
              }
            }
          }
        }
      },
      "post": {
        "tags": [
          "Esta página describe la API del servicio creado"
        ],
        "operationId": "enviarMensaje",
        "requestBody": {
          "content": {
            "**/*": {
              "schema": {
                "type": "string"
              }
            }
          }
        },
        "responses": {
          "default": {
            "description": "default response",
            "content": {
              "application/json": {
                "schema": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    },
    "/api/contador": {
      "put": {
        "operationId": "inicializar",
        "responses": {
          "default": {
            "description": "default response",
            "content": {
              "application/json": {
                "schema": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

3) Seguido de esto, invocamos la pagina de informacion de swagger, quedando de la siguiente manera:



Desarrollo de API RESTFul

1) Primero creamos un nuevo proyecto, donde desarrollaremos cada clase. Creamos el proyecto MVN y dentro del mismo primero crearemos empleado:



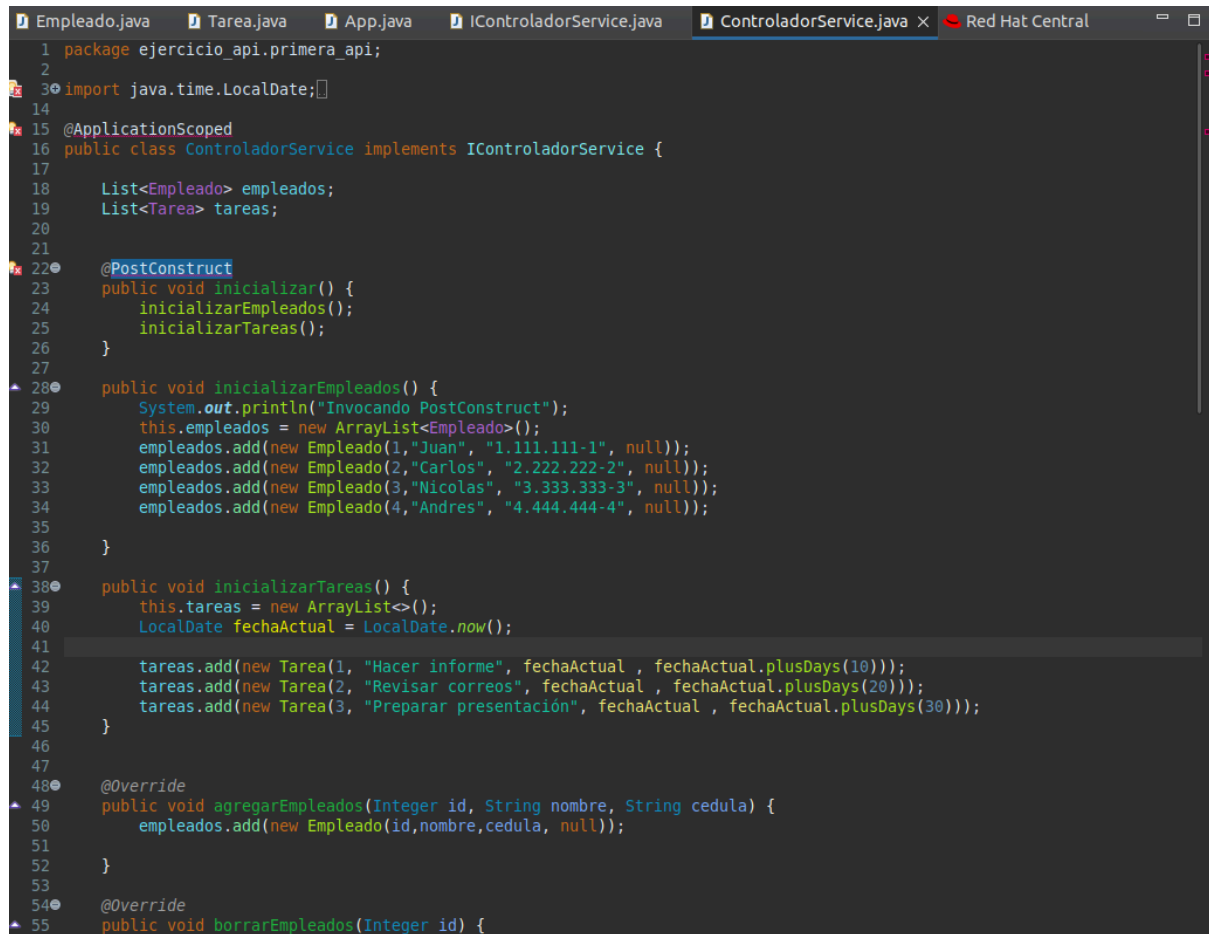
Y de la misma manera procedemos a crear la clase Tarea:

```
Empleado.java  Tarea.java x  App.java
1 package ejercicio_api.primer_api;
2
3 import java.time.LocalDate;
4
5 public class Tarea {
6
7     private Integer id;
8     private String descripcion;
9     private LocalDate fechaInicio;
10    private LocalDate fechaFin;
11
12    public Tarea() {
13
14    }
15
16    public Tarea(Integer id, String descripcion, LocalDate fechaInicio, LocalDate fechaFin) {
17        this.id = id;
18        this.descripcion = descripcion;
19        this.fechaInicio = fechaInicio;
20        this.fechaFin = fechaFin;
21    }
22
23    public Tarea(Integer id) {
24        this.id = id;
25    }
26
27    public Integer getId() {
28        return id;
29    }
30    public void setId(Integer id) {
31        this.id = id;
32    }
33
34    public String getDescripcion() {
35        return descripcion;
36    }
37    public void setDescripcion(String descripcion) {
38        this.descripcion = descripcion;
39    }
40
41 }
```


Luego de esto, procederemos a crear la interfaz Controlador Service (IControladorService) con sus notificaciones respectivas.

```
Empleado.java  Tarea.java  App.java  IControladorService.java x  ControladorService.java  Red Hat Central
1 package ejercicio_api.primer_api;
2
3 import java.util.List;
4
5
6
7
8 public interface IControladorService {
9     /**
10      * @param id -> Identificador del empleado a crear
11      * @param nombre -> Nombre del empleado a crear
12      * @param cedula -> cedula del empleado a crear
13      */
14     public void agregarEmpleados(Integer id, String nombre, String cedula);
15
16
17     /**
18      * @param id -> id del empleado a borrar
19      */
20     public void borrarEmpleados(Integer id);
21
22     /**
23      * @return devuelve una lista con los datos de todos los empleados
24      */
25     public List<Empleado> listarEmpleados();
26
27     /**
28      * @return devuelve una lista con los datos de todas las tareas.
29      */
30     public List<Tarea> listarTareas();
31
32     /**
33      * @param id -> id del empleado a buscar
34      * @return Devuelve el objeto del empleado buscado.
35      */
36     public Empleado obtenerEmpleado(Integer id);
37
38     /**
39      * @param idEmpleado -> id del empleado
40      * @param idTarea -> id de la tarea a asignar
41      */
42     public void agregarTareaEmpleado(Integer idEmpleado, Integer idTarea);
43
44     /**
45      * @param idEmpleado -> id del empleado
46      * @return devuelve una lista de tareas de dicho empleado
47      */
48     public List<Tarea> listaTareasEmpleado(Integer idEmpleado);
49 }
```

Y también luego su implementación

A screenshot of an IDE window showing the implementation of the ControladorService class. The window has several tabs at the top: Empleado.java, Tarea.java, App.java, IControladorService.java, ControladorService.java (selected), and Red Hat Central. The code is in Java and implements the IControladorService interface. It includes package and import statements, class annotations, and several methods for initializing and managing employees and tasks.

```
1 package ejercicio_api.primer_api;
2
3 import java.time.LocalDate;
4
15 @ApplicationScoped
16 public class ControladorService implements IControladorService {
17
18     List<Empleado> empleados;
19     List<Tarea> tareas;
20
21
22     @PostConstruct
23     public void inicializar() {
24         inicializarEmpleados();
25         inicializarTareas();
26     }
27
28     public void inicializarEmpleados() {
29         System.out.println("Invocando PostConstruct");
30         this.empleados = new ArrayList<Empleado>();
31         empleados.add(new Empleado(1, "Juan", "1.111.111-1", null));
32         empleados.add(new Empleado(2, "Carlos", "2.222.222-2", null));
33         empleados.add(new Empleado(3, "Nicolas", "3.333.333-3", null));
34         empleados.add(new Empleado(4, "Andres", "4.444.444-4", null));
35     }
36
37
38     public void inicializarTareas() {
39         this.tareas = new ArrayList<>();
40         LocalDate fechaActual = LocalDate.now();
41
42         tareas.add(new Tarea(1, "Hacer informe", fechaActual, fechaActual.plusDays(10)));
43         tareas.add(new Tarea(2, "Revisar correos", fechaActual, fechaActual.plusDays(20)));
44         tareas.add(new Tarea(3, "Preparar presentación", fechaActual, fechaActual.plusDays(30)));
45     }
46
47
48     @Override
49     public void agregarEmpleados(Integer id, String nombre, String cedula) {
50         empleados.add(new Empleado(id, nombre, cedula, null));
51     }
52
53
54     @Override
55     public void borrarEmpleados(Integer id) {
```

Luego de esto podemos asentarnos en implementar los casos de uso para implementar los test.

Todo el código puede ser visto en el repositorio de GitHub:

https://github.com/JuanmaPilon/JavaEE___Practices