El futuro digital es de todos

MinTIC

SENA

Servicio Nacional de Aprendizaje SENA

INTEROPERABILIDAD DE SISTEMAS DE INFORMACIÓN MEDIANTE X-ROAD

RECURSOS PRÁCTICOS DE APOYO

Guía de Implementación

Esta guía se desarrolla teniendo en cuenta los lineamientos del Marco de interoperabilidad para Gobierno Digital de Agosto de 2019, definido por MinTIC, sección 4.4 Dominio Técnico.

Como caso de estudio se realizará la implementación de un sistema de software distribuido a partir de la construcción de dos servicios web, uno de tipo REST, otro de tipo SOAP, y su respectivo componente de base de datos. Este ejercicio se desarrollará mediante la tecnología Spring Boot, la cual está basada en el lenguaje de programación Java.

Requisitos

1. **Editor de texto o IDE**

    a. IntelliJ IDEA (usado en este ejercicio).
    b. Spring Tools 4

2. **Java**

    Descargar e instalar Java Development Kit (JDK) según el sistema operativo (el ejercicio se desarrollará con Java 8).

3. **MAVEN**

    Descargar e instalar Apache Maven según el sistema operativo.

4. **PostgreSQL**

    a. Descargar e instalar PostgreSQL según el sistema operativo.
    b. Utilizar el gestor de bases de datos de preferencia, se recomienda pgAdmin 4.

5. **POSTMAN**

    Descargar e instalar Postman según el sistema operativo.

6. **SOAPUI**

    Descargar e instalar SoapUI según el sistema operativo.

7. **Docker**

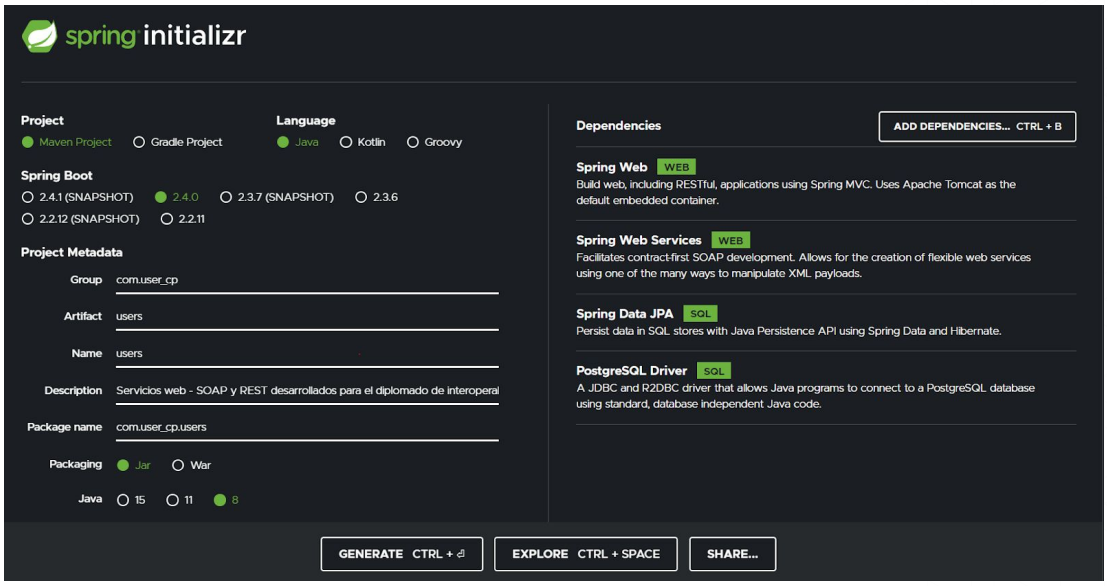
    Instalar y ejecutar **Docker**, siguiendo la **Guía de Instalación de Docker** (*Sección 2.3*).

Desarrollo

1. Ingresar a Spring Initializr y generar el esqueleto del sistema de software, debe adicionar las siguientes dependencias:

    a. Spring Web
    b. Spring Web Services
    c. Spring Data JPA
    d. PostgreSQL Driver

    A continuación se presenta la configuración inicial utilizada para el desarrollo del escenario:



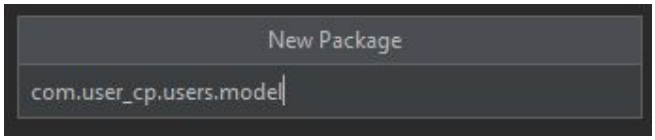
2. Presionar el botón **Generate**, descargar el código y descomprimirlo. Este será el código base para este ejercicio. Abrirlo mediante el editor de preferencia.

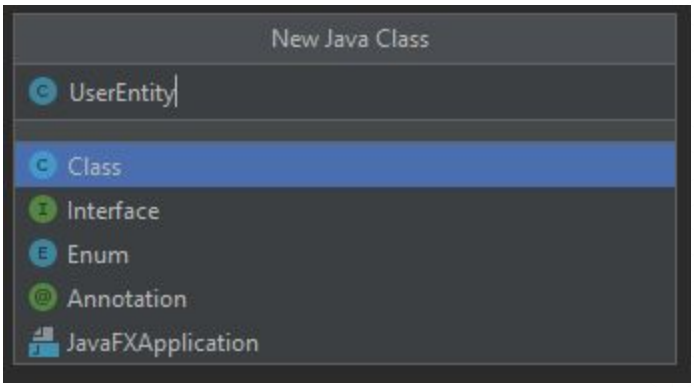
3. Siguiendo la arquitectura de la Sección 3.3, primero se debe realizar la configuración de la **base de datos**, además de las conexiones para el servidor y la comunicación entre estos. Para lo cual, dirigirse al archivo ***users/src/main/resources/application.properties*** y agregar el siguiente código:

```
server.port=8080

spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://${USERS_DB_DNS:localhost}/${US
ERS_DB_NAME:users}
spring.datasource.username=${USERS_DB_USER:postgres}.
```

```
spring.datasource.password=${USERS_DB_PASSWORD:postgres}

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreS
QLDialect
spring.jpa.properties.hibernate.jdbc.time_zone=America/Bogota.
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

4. Luego, en cuanto al **componente lógico** del sistema, se debe realizar la **Capa Modelo**, para esto, bajo el paquete principal, crear uno llamado ***model*** y dentro de este, la clase ***UserEntity***:





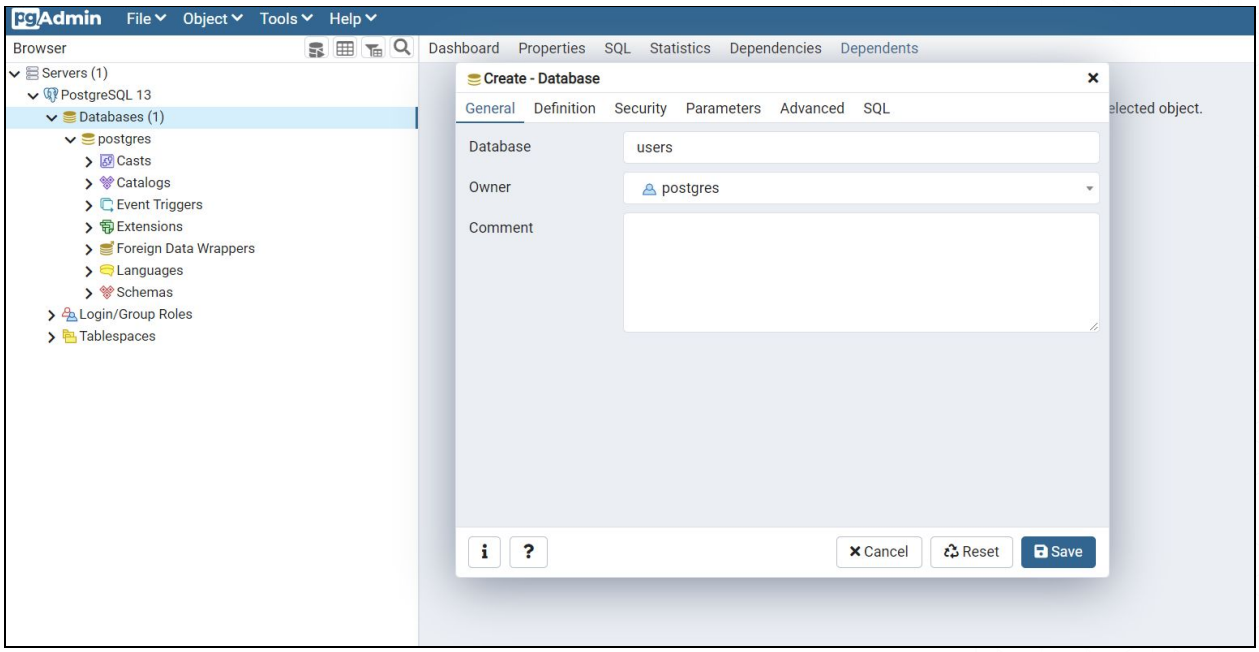
5. Dentro de dicha clase se crearán las representaciones de los datos sobre la base de datos. Para este ejercicio solo se tendrá una entidad, donde se definen sus atributos, uno o varios constructores según se requiera y los *getters* y *setters* respectivos. Añadir el siguiente código:

```
package com.user_cp.users.model;

import javax.persistence.*;

@Entity
@Table(name = "users")
public class UserEntity {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Integer id;
   private String firstname;
   private String lastname;
   private String address;
```

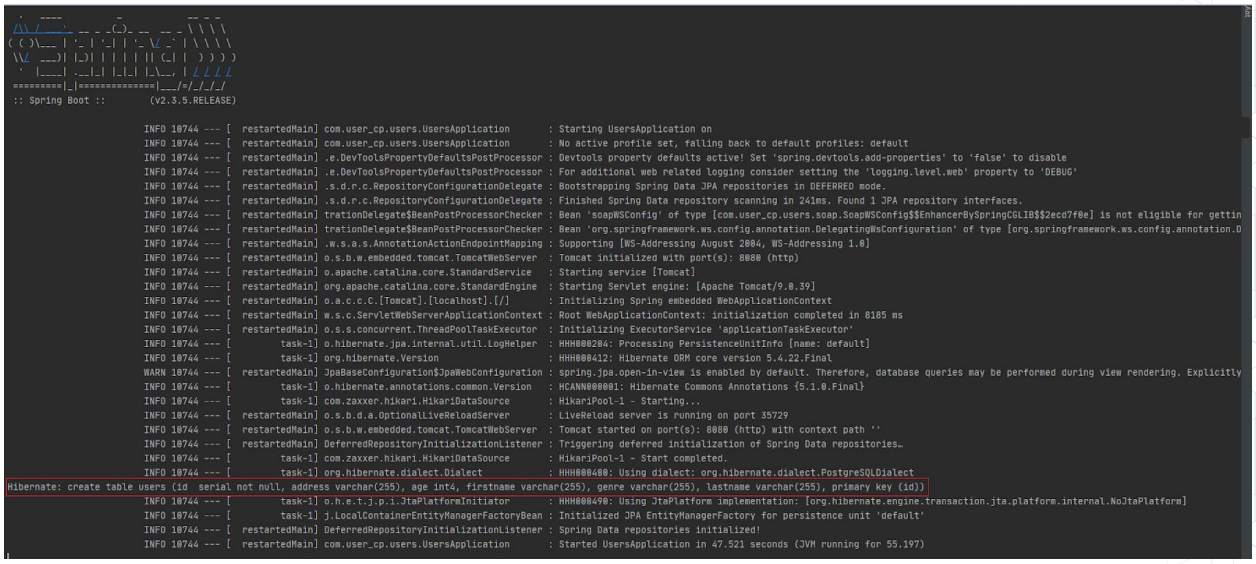
```
    private String genre;
    private Integer age;

    public UserEntity(){

    }

        public UserEntity(String firstname, String lastname, String
address, String genre, Integer age) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.address = address;
        this.genre = genre;
        this.age = age;
    }

     public UserEntity(Integer id, String firstname, String lastname,
String address, String genre, Integer age) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.address = address;
        this.genre = genre;
        this.age = age;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }
```

```
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

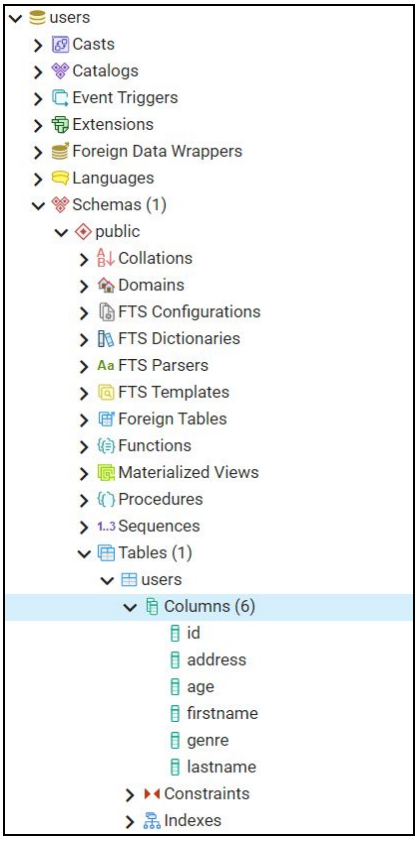
6. En este punto, se puede probar la conexión con la base de datos, para lo cual se debe ingresar al gestor, por ejemplo *pgAdmin 4*. Se debe contar con un usuario llamado ***postgres*** que tenga ***postgres*** como contraseña y se debe crear una base de datos llamada ***users***.



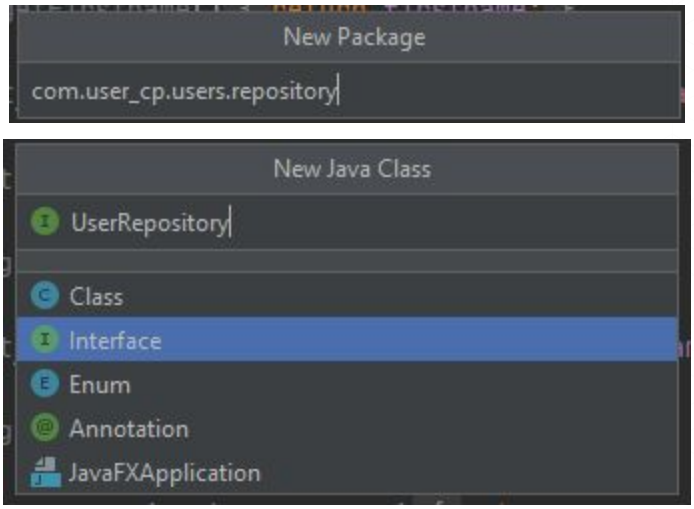
7. Ahora, para probar la conexión, se debe hacer ***build*** en el proyecto y ***ejecutarlo***, se obtendrá el siguiente resultado:



8. Dirigirse nuevamente al gestor de base de datos y verificar la creación de la tabla ***users*** y sus respectivas columnas.



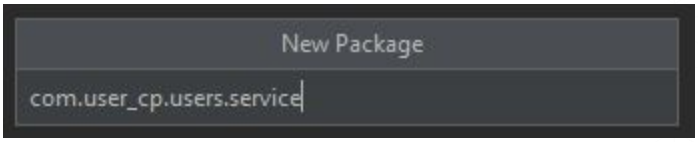
9. Luego, en cuanto al **componente lógico** del sistema, se debe realizar la **Capa Repositorio**, para esto, bajo el paquete principal, crear uno llamado ***repository*** y dentro de este, la interfaz ***UserRepository***:

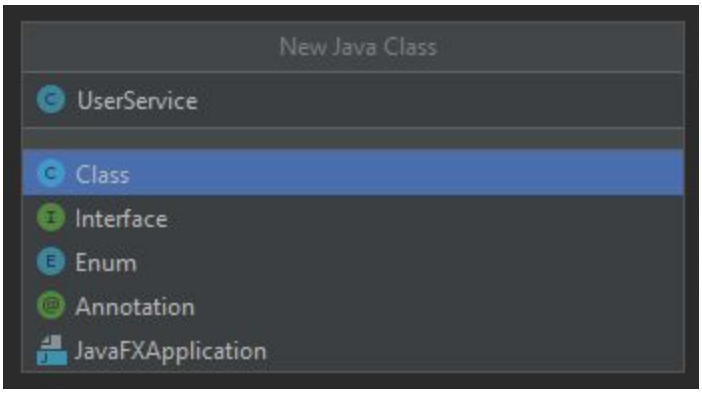


10. Esta interfaz extiende del ***JpaRepository*** sobre una entidad específica, se debe agregar el siguiente código:

```
package com.user_cp.users.repository;

import com.user_cp.users.model.UserEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<UserEntity,
Integer> {

}
```

11. En cuanto al **componente lógico** del sistema, se debe realizar la **Capa Servicio**, para esto, bajo el paquete principal, crear uno llamado ***service*** y dentro de este, la clase ***UserService***:



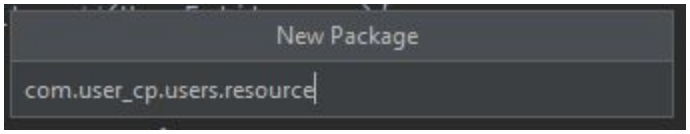


12. Esta clase instancia ***userRepository*** para que, a partir de este, se puedan crear diferentes clases que manejen el CRUD de la entidad, se debe agregar el siguiente código:

```
package com.user_cp.users.service;

import com.user_cp.users.model.UserEntity;
import com.user_cp.users.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public UserEntity add(UserEntity user){
        try{
            return userRepository.save(user);
        }catch (Exception e){
            e.printStackTrace();
            return null;
        }
    }

    public List<UserEntity> getAll(){
        return userRepository.findAll();
    }

    public UserEntity getByID(Integer id){
        if(userRepository.existsById(id)){
            return userRepository.findById(id).get();
```

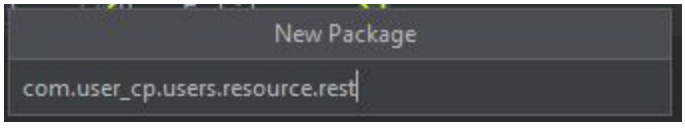
```
        }
        return null;
    }

    public UserEntity update(UserEntity user){
        Integer id = user.getId();
        if(userRepository.existsById(id)){
            return userRepository.save(user);
        }
        return null;
    }

    public Boolean delete(Integer id){
        if(userRepository.existsById(id)) {
            userRepository.deleteById(id);
            return true;
        }
        return false;
    }
}
```

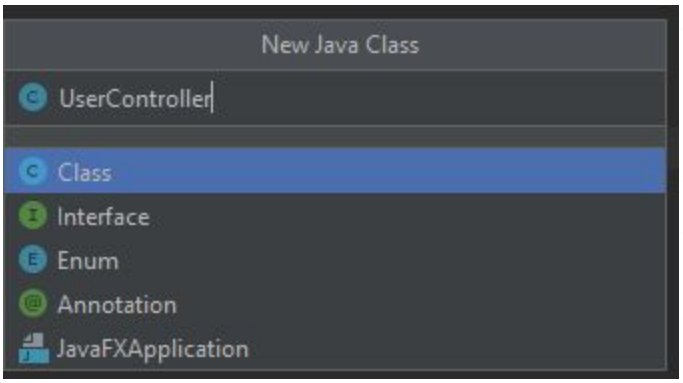
13. En cuanto al **componente lógico** del sistema, se debe agregar la **Capa Recurso**, para esto, bajo el paquete principal, crear uno llamado ***resource***.



REST

1. En cuanto al **componente lógico** del sistema, se debe realizar el **Segmento de Capa REST**, para esto, bajo el paquete ***resource***, crear uno llamado ***rest*** y dentro de este, la clase ***UserController***:





2. Esta clase instancia ***userService***, con lo cual, se mapean los métodos **HTTP** desde solicitudes de servidor y se da manejo a las respuestas, se debe agregar el siguiente código:

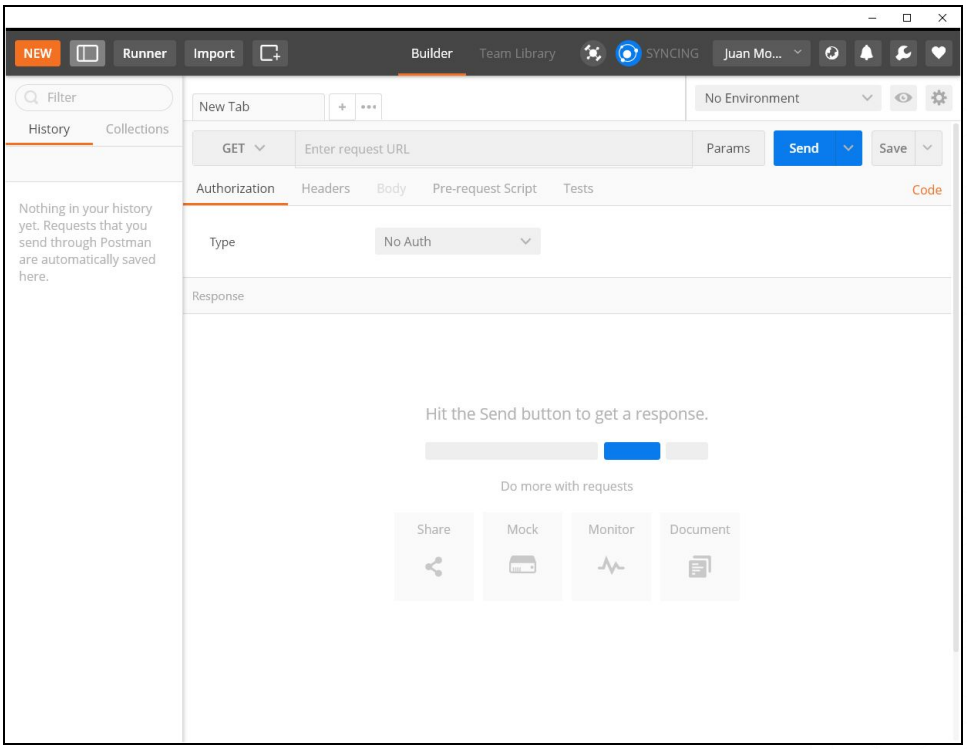
```
package com.user_cp.users.resource.rest;

import com.user_cp.users.model.UserEntity;
import com.user_cp.users.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class UserController {

   @Autowired
   private UserService userService;

   @GetMapping("/users")
   public List<UserEntity> getAllUsers(){
       return userService.getAll();
   }

   @GetMapping("/users/{id}")
     public  ResponseEntity<UserEntity>  getUserById(@PathVariable("id")
Integer id){
       UserEntity user = userService.getByID(id);
       return user == null ?
               new ResponseEntity<>(null, HttpStatus.NOT_FOUND)  :
               new ResponseEntity<>(user, HttpStatus.OK);
   }
```

```
    @PostMapping("/users")
    public UserEntity addUser(@RequestBody UserEntity user){
        userService.add(user);
        return user;
    }

    @PutMapping("/users")
          public   ResponseEntity<UserEntity>   updateUser(@RequestBody
UserEntity user){
        UserEntity aux = userService.update(user);
        return aux == null ?
                new ResponseEntity<>(null, HttpStatus.NOT_FOUND) :
                new ResponseEntity<>(aux, HttpStatus.OK);
    }

    @DeleteMapping("/users/{id}")
        public  ResponseEntity<Boolean>  deleteUser(@PathVariable("id")
Integer id){
        Boolean flag = userService.delete(id);
        return !flag ?
                new ResponseEntity<>(false, HttpStatus.NOT_FOUND) :
                new ResponseEntity<>(true, HttpStatus.OK);
    }
}
```

3. En este punto, ya se ha creado el *servicio Rest,* para probarlo, se debe hacer ***build*** del proyecto y se debe ***ejecutar*** (mantenerlo en ejecución). Verificar que el servicio se inició correctamente, ingresando a ***http://localhost:8080/*** mediante el navegador:


```
INFO 13844 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
INFO 13844 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
INFO 13844 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
```

4. Si se desea, se puede verificar el estado de la tabla ***users***, tras la ejecución de cada una de las peticiones que se realizarán e igualmente la terminal, finalmente, se debe abrir ***Postman***.

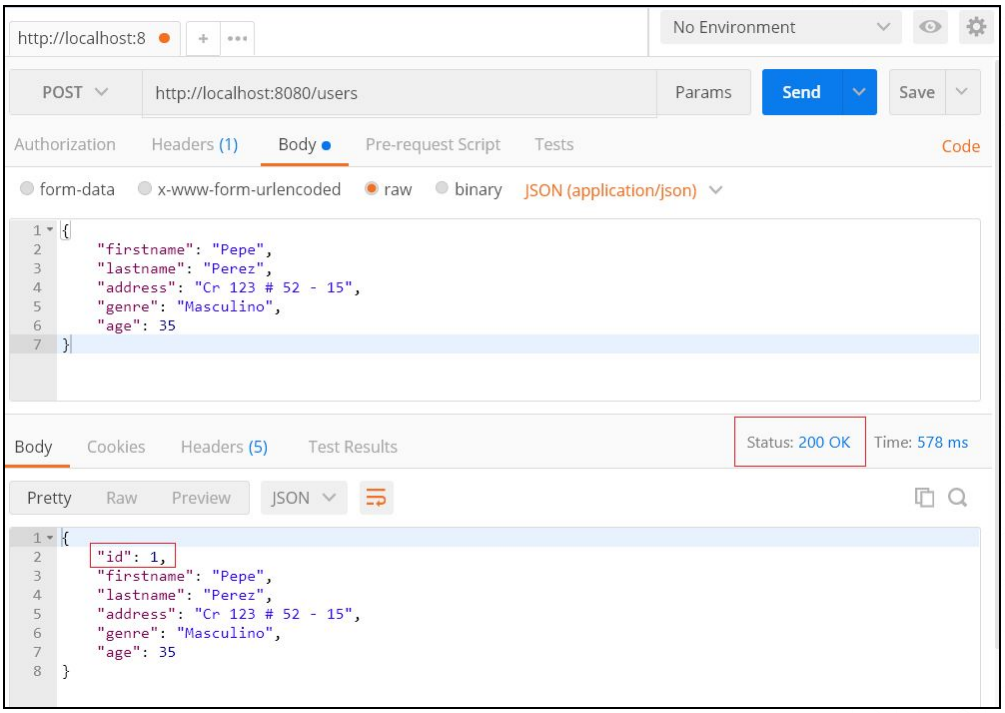


5. Para **crear** un nuevo usuario, configurar la petición de la siguiente manera:

   - Método HTTP (en la lista desplegable): POST
   - Request URL (en la entrada de texto junto al método): http://localhost:8080/users
   - Cuerpo de la petición (Pestaña Body → Seleccionar la opción “raw” y en el desplegable que aparece, cambiar “Text” por “JSON”), pegar el siguiente objeto JSON

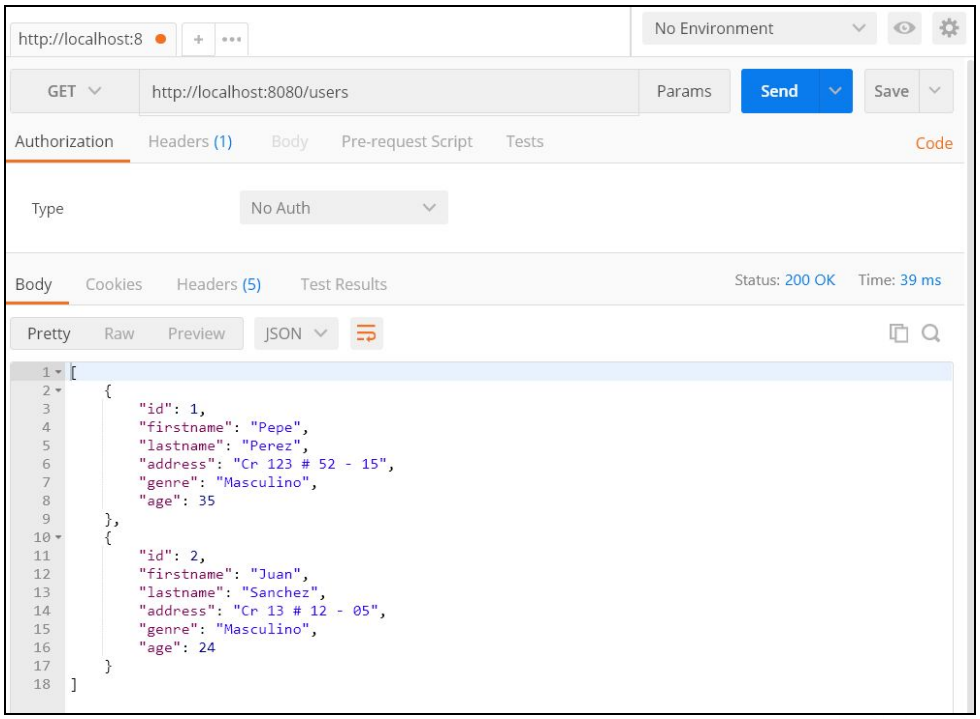
```
{
	"firstname": "Pepe",
	"lastname": "Perez",
	"address": "Cr 123 # 52 - 15",
	"genre": "Masculino",
	"age": 35
}
```

6. Hacer clic en el botón “Send” para solicitar al servidor ese recurso:



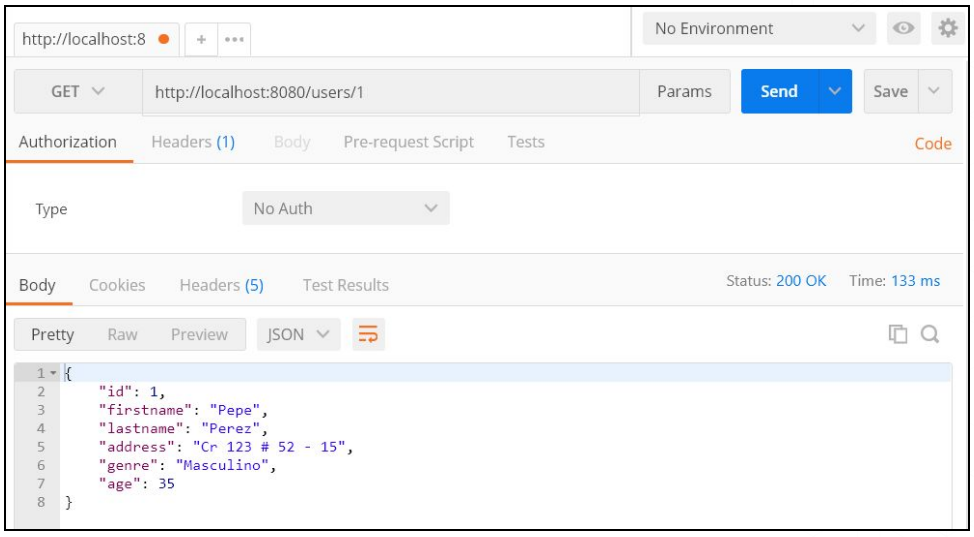
7. Para **obtener** todos los usuarios. Configurar la petición de la siguiente manera:

    - Método HTTP (en la lista desplegable): GET
    - Request URL (en la entrada de texto junto al método): http://localhost:8080/users



8. Para **obtener** un usuario mediante su **Id**, configurar la petición de la siguiente manera:

    - Método HTTP (en la lista desplegable): GET
    - Request URL (en la entrada de texto junto al método): http://localhost:8080/users/{id}



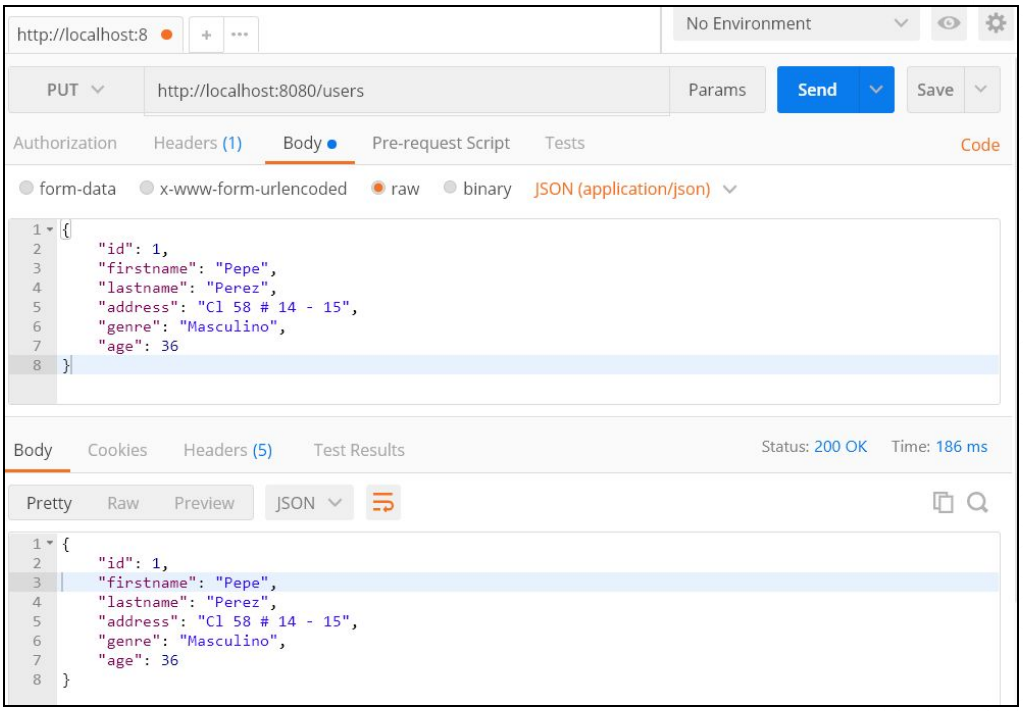
9. Para **modificar** un usuario, configurar la petición de la siguiente manera:

    - Método HTTP (en la lista desplegable): PUT

- Request URL (en la entrada de texto junto al método): http://localhost:8080/users
- Cuerpo de la petición (Pestaña Body → Seleccionar la opción “raw” y en el desplegable que aparece cambiar “Text” por “JSON”), pegar el siguiente objeto JSON

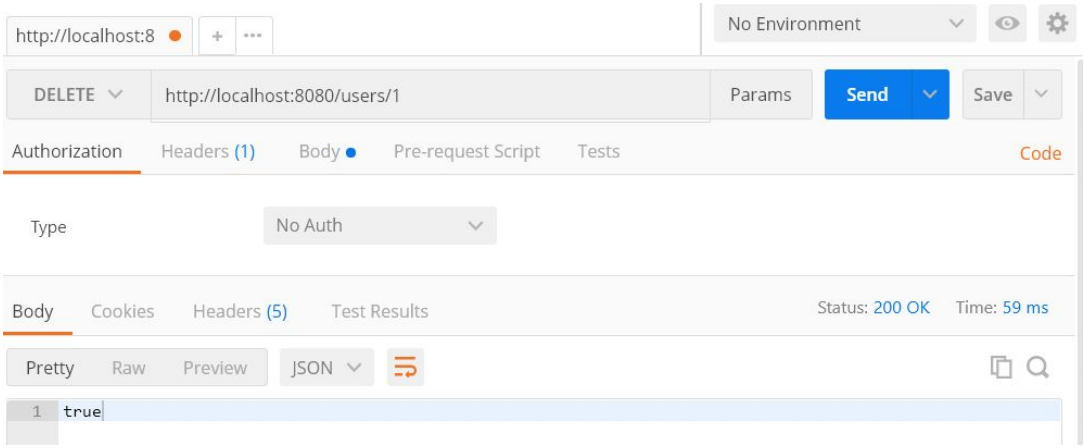
```
{
    "id": 1,
    "firstname": "Pepe",
    "lastname": "Perez",
    "address": "Cl 58 # 14 - 15",
    "genre": "Masculino",
    "age": 36
}
```

10. Hacer clic en el botón “Send” para solicitar al servidor el recurso:



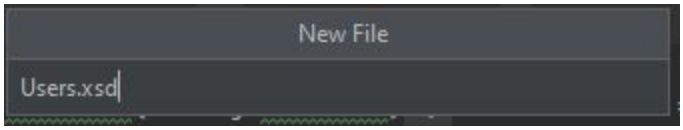
11. Para **eliminar** un usuario mediante su **Id**, configurar la petición de la siguiente manera:

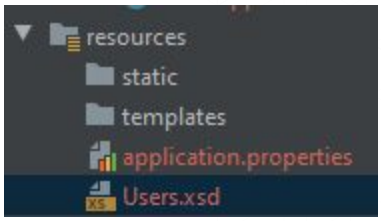
- Método HTTP (en la lista desplegable): DELETE
- Request URL (en la entrada de texto junto al método): http://localhost:8080/users/{id}



SOAP

1. En cuanto al **componente lógico** del sistema, se debe realizar el **Segmento de Capa SOAP**, para esto, en primera instancia, se debe crear el archivo ***Users.xsd*** bajo la ruta ***users/src/main/resources/***:





2. A continuación se presenta el ***esquema XSD***, le cual permite crear las funciones y entidades a trabajar mediante el servicio SOAP, se debe agregar el siguiente código:

```
<xs:schema                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://localhost/users"
                              targetNamespace="http://localhost/users"
elementFormDefault="qualified">

   <xs:element name="getUserByIdRequest">
       <xs:complexType>
           <xs:sequence>
               <xs:element name="userId" type="xs:int"/>
           </xs:sequence>
       </xs:complexType>
   </xs:element>
```

```
    <xs:element name="getUserByIdResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="user" type="tns:user"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="user">
        <xs:sequence>
            <xs:element name="userId" type="xs:int" />
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="genre" type="xs:string"/>
            <xs:element name="age" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="getAllUsersRequest">
        <xs:complexType/>
    </xs:element>
    <xs:element name="getAllUsersResponse">
        <xs:complexType>
            <xs:sequence>
                                <xs:element  name="user"  type="tns:user"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="serviceStatus">
        <xs:sequence>
            <xs:element name="statusCode" type="xs:string"/>
            <xs:element name="message" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="addUserRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                <xs:element name="address" type="xs:string"/>
                <xs:element name="genre" type="xs:string"/>
                <xs:element name="age" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
    <xs:element name="addUserResponse">
        <xs:complexType>
            <xs:sequence>
                                        <xs:element   name="serviceStatus"
type="tns:serviceStatus"/>
                <xs:element name="user" type="tns:user"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="updateUserRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="userId" type="xs:int" />
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                <xs:element name="address" type="xs:string"/>
                <xs:element name="genre" type="xs:string"/>
                <xs:element name="age" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="updateUserResponse">
        <xs:complexType>
            <xs:sequence>
                                        <xs:element   name="serviceStatus"
type="tns:serviceStatus"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="deleteUserRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="userId" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="deleteUserResponse">
        <xs:complexType>
            <xs:sequence>
                                        <xs:element   name="serviceStatus"
type="tns:serviceStatus"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
</xs:schema>
```

3. Para que el servicio SOAP funcione, primero se debe agregar la siguiente dependencia al archivo ***pom.xml*** en la sección ***dependencies***:

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

4. También, sobre el mismo archivo, en la sección ***build - plugins*** se debe agregar el siguiente plugin que permitirá crear mediante el archivo ***Users.xsd*** la lógica de transferencia a Java. **Nota:** Asegurarse de que la dirección de la etiqueta ***sources*** dirija correctamente al archivo ***Users.xsd***.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>2.5.0</version>
  <executions>
     <execution>
        <id>xjc</id>
        <goals>
           <goal>xjc</goal>
        </goals>
     </execution>
  </executions>
  <configuration>
     <sources>
        <source>src/main/resources/Users.xsd</source>
     </sources>
  </configuration>
</plugin>
```

5. Finalmente, el archivo de configuración ***pom.xml*** quedará de la siguiente manera:

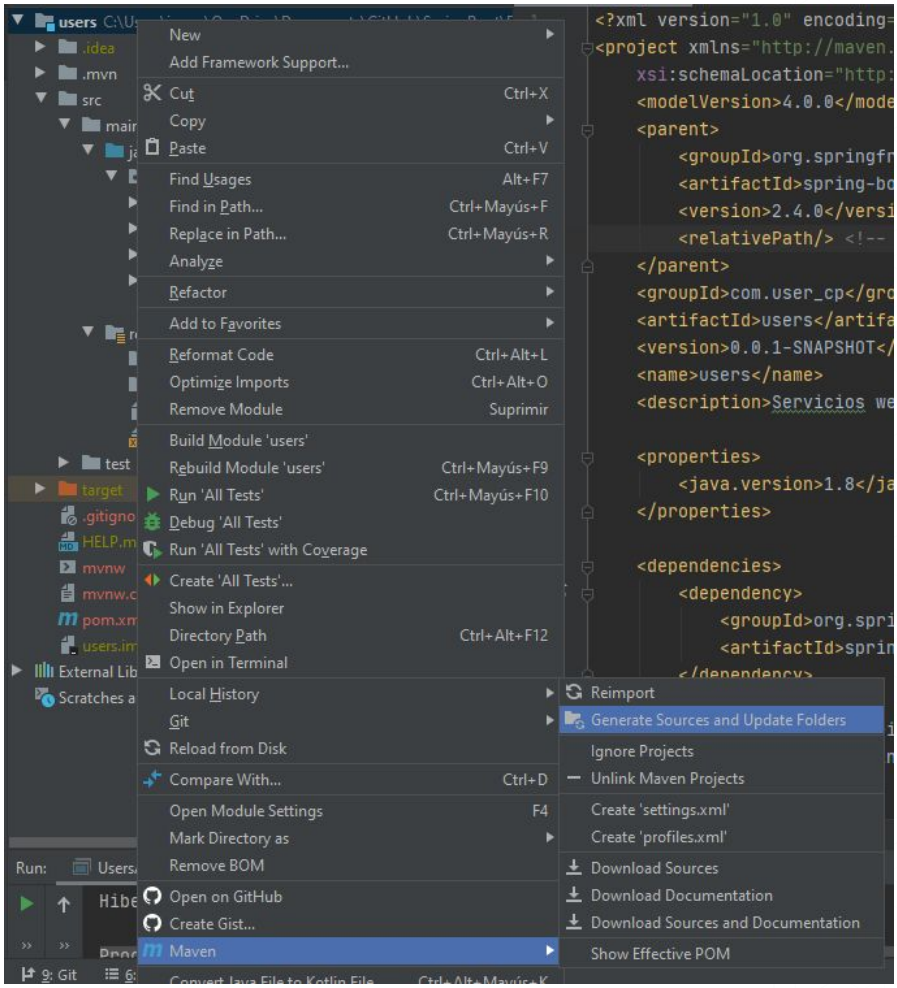
```
<?xml version="1.0" encoding="UTF-8"?>
<project                        xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
     <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.user_cp</groupId>
  <artifactId>users</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>users</name>
   <description>Servicios web - SOAP y REST desarrollados para el
diplomado de interoperabilidad con XRoad</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

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

    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
```

```
    <plugins>
       <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
       </plugin>
       <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>jaxb2-maven-plugin</artifactId>
          <version>2.5.0</version>
          <executions>
             <execution>
                <id>xjc</id>
                <goals>
                   <goal>xjc</goal>
                </goals>
             </execution>
          </executions>
          <configuration>
             <sources>
                <source>src/main/resources/Users.xsd</source>
             </sources>
          </configuration>
       </plugin>
    </plugins>
  </build>

</project>
```

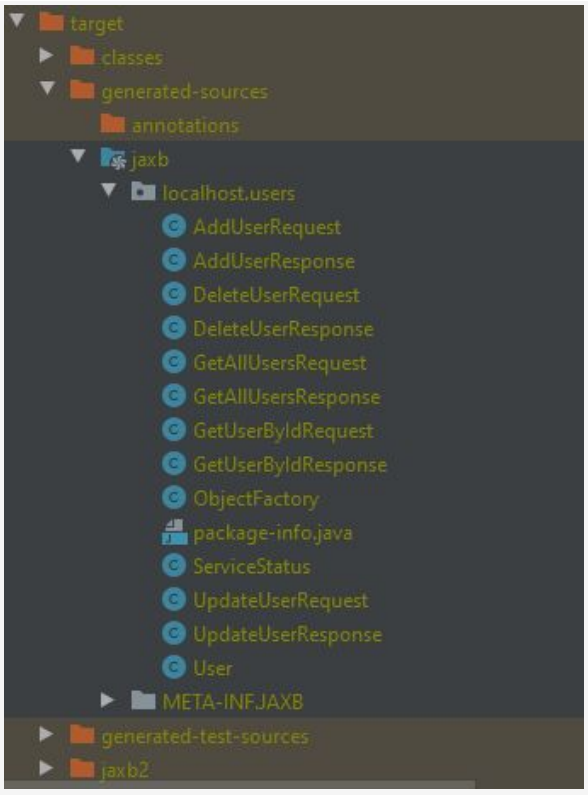
6. Asegurarse de cargar los cambios a ***Maven***.

7. Para poder utilizar las funciones y entidades establecidas por el XSD mediante lógica de Java, se deberá generar el código fuente, en **IntelliJ IDEA** se realiza haciendo clic derecho sobre la carpeta del proyecto y luego *Maven - Generate Sources and Update Folders*:

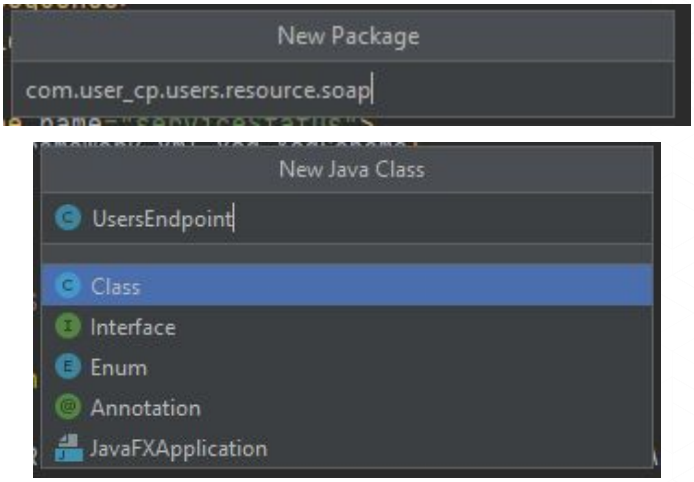


8. Las clases de Java se generarán en la carpeta ***target/generate-sources/jaxb/<package-name>.<package-name>*** el cual se especifica en el archivo ***Users.xsd***. Alternativamente, ejecutando el siguiente comando en el directorio principal:

```
mvn package
```



9. Finalmente, para completar el **Segmento de Capa REST**, se deben crear dos clases: en el paquete ***resource***, crear uno llamado ***soap*** y dentro de este, la clase ***UsersEndpoint***:



10. Esta clase maneja las solicitudes y respuestas de las peticiones ***SOAP***, interactuando entre el código fuente generado anteriormente mediante *Maven* y las capas modelo, repositorio y servicio. Se debe agregar el siguiente código:

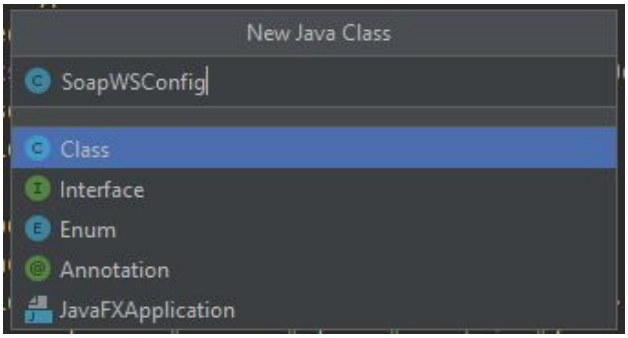
```
package com.user_cp.users.resource.soap;

import java.util.ArrayList;
import java.util.List;

import com.user_cp.users.model.UserEntity;
import com.user_cp.users.service.UserService;
import localhost.users.*;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
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;


@Endpoint
public class UsersEndpoint {

            public    static    final    String    NAMESPACE_URI    =
"http://localhost/users";

   private UserService userService;

   public UsersEndpoint(){

   }

   @Autowired
   public UsersEndpoint(UserService userService) {
       this.userService = userService;
   }

         @PayloadRoot(   namespace   =   NAMESPACE_URI,   localPart   =
"getUserByIdRequest" )
   @ResponsePayload
           public    GetUserByIdResponse    getUserById(@RequestPayload
GetUserByIdRequest request){
       GetUserByIdResponse response = new GetUserByIdResponse();
                                       UserEntity     userEntity     =
userService.getByID(request.getUserId());
       User user = new User();
       user.setUserId(userEntity.getId());
       BeanUtils.copyProperties(userEntity, user);
```

```
        response.setUser(user);
        return response;
    }

        @PayloadRoot(  namespace  =  NAMESPACE_URI,  localPart  =
"getAllUsersRequest")
    @ResponsePayload
            public   GetAllUsersResponse   getAllUsers(@RequestPayload
GetAllUsersRequest request){
        GetAllUsersResponse response = new GetAllUsersResponse();
        List<User> userList = new ArrayList<User>();
        List<UserEntity> userEntityList = userService.getAll();
        for (UserEntity userEntity : userEntityList){
            User user = new User();
            user.setUserId(userEntity.getId());
            BeanUtils.copyProperties(userEntity, user);
            userList.add(user);
        }
        response.getUser().addAll(userList);
        return response;
    }

        @PayloadRoot(  namespace  =  NAMESPACE_URI,  localPart  =
"addUserRequest")
    @ResponsePayload
      public  AddUserResponse  addUser(@RequestPayload  AddUserRequest
request) {
        AddUserResponse response = new AddUserResponse();
        User newUser = new User();
        ServiceStatus serviceStatus = new ServiceStatus();

                                UserEntity    newUserEntity    =    new
UserEntity(request.getFirstname(),              request.getLastname(),
request.getAddress(), request.getGenre(), request.getAge());
        UserEntity savedUserEntity = userService.add(newUserEntity);

        if (savedUserEntity == null) {
            serviceStatus.setStatusCode("CONFLICT");
            serviceStatus.setMessage("Error adding user");
        } else {
            newUser.setUserId(savedUserEntity.getId());
            BeanUtils.copyProperties(savedUserEntity, newUser);
            serviceStatus.setStatusCode("SUCCESS");
            serviceStatus.setMessage("User adding succesfully");
        }
```

```
        response.setUser(newUser);
        response.setServiceStatus(serviceStatus);
        return response;
    }

           @PayloadRoot(namespace    =    NAMESPACE_URI,    localPart    =
"updateUserRequest")
    @ResponsePayload
             public    UpdateUserResponse    updateUser(@RequestPayload
UpdateUserRequest request) {
        UpdateUserResponse response = new UpdateUserResponse();
        ServiceStatus serviceStatus = new ServiceStatus();

                                    UserEntity    userEntity    =    new
UserEntity(request.getUserId(),request.getFirstname(),
request.getLastname(),    request.getAddress(),    request.getGenre(),
request.getAge());
        UserEntity updatedUser = userService.update(userEntity);

        if(updatedUser == null) {
            serviceStatus.setStatusCode("NOT FOUND");
             serviceStatus.setMessage("User = " + request.getFirstname()
+ " " + request.getLastname() + " doesn't exist");
        }else {
            serviceStatus.setStatusCode("SUCCESS");
            serviceStatus.setMessage("User update succesfully");
        }

        response.setServiceStatus(serviceStatus);
        return response;
    }

           @PayloadRoot(namespace    =    NAMESPACE_URI,    localPart    =
"deleteUserRequest")
    @ResponsePayload
             public    DeleteUserResponse    deleteUser(@RequestPayload
DeleteUserRequest request) {
        DeleteUserResponse response = new DeleteUserResponse();
        ServiceStatus serviceStatus = new ServiceStatus();

        boolean flag = userService.delete(request.getUserId());

        if (!flag) {
            serviceStatus.setStatusCode("FAIL");
             serviceStatus.setMessage("Error deleting user whit id = " +
request.getUserId());
```

```
        } else {
            serviceStatus.setStatusCode("SUCCESS");
            serviceStatus.setMessage("User delete succesfully");
        }

        response.setServiceStatus(serviceStatus);
        return response;
    }
}
```

11. Finalmente, se debe crear la clase ***SoapWSConfig*** en el paquete ***soap***.

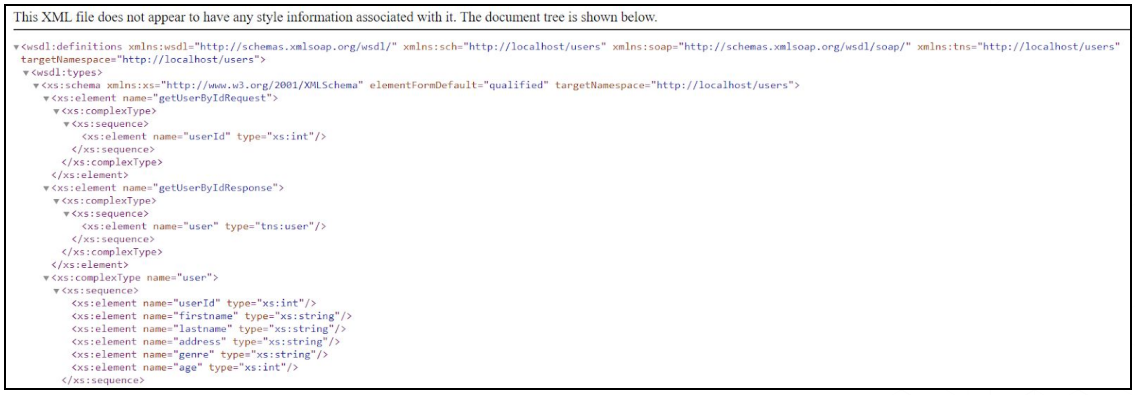


12. Esta clase configura el servicio SOAP y la comunicación mediante ***WSDL***, se debe agregar el siguiente código:

```
package com.user_cp.users.resource.soap;

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.WsConfigurerAdapter;
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 SoapWSConfig extends WsConfigurerAdapter {

   @SuppressWarnings({ "rawtypes", "unchecked" })
```

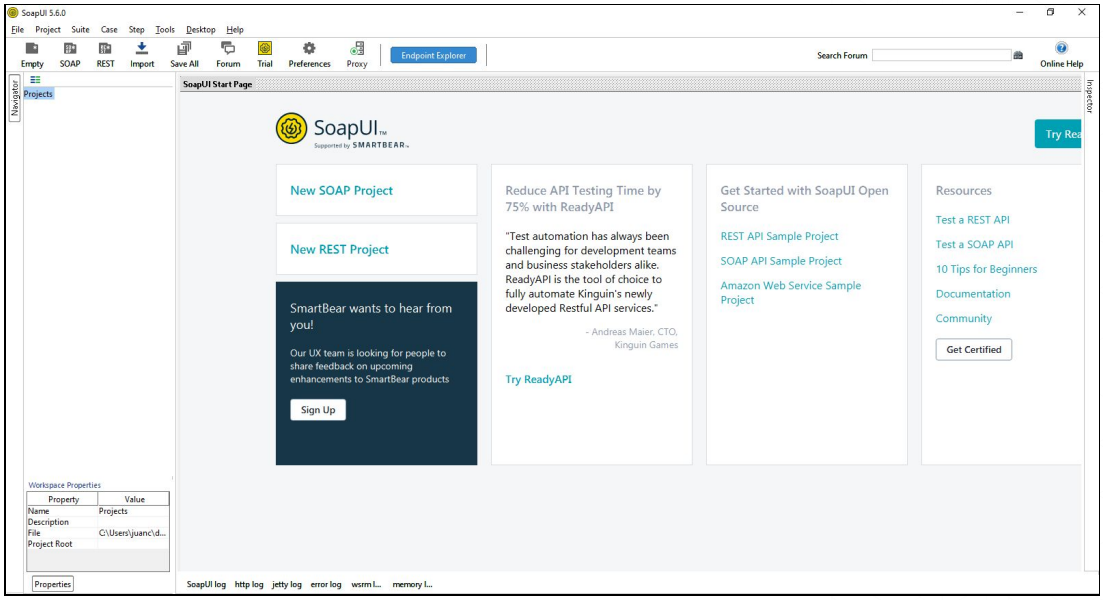
```java
    @Bean
                                public          ServletRegistrationBean
messageDispatcherServlet(ApplicationContext appContext){
                            MessageDispatcherServlet    servlet    =    new
MessageDispatcherServlet();
        servlet.setApplicationContext(appContext);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

    @Bean(name = "users")
      public  DefaultWsdl11Definition  defaultWsdl11Definition(XsdSchema
schema){
                      DefaultWsdl11Definition   wsdl11Definition   =   new
DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("UsersPort");
        wsdl11Definition.setLocationUri("/ws");

wsdl11Definition.setTargetNamespace(UsersEndpoint.NAMESPACE_URI);
        wsdl11Definition.setSchema(schema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema usuariosSchema(){
        return new SimpleXsdSchema(new ClassPathResource("Users.xsd"));
    }

}
```

13. En este punto, ya se ha creado el *servicio SOAP,* para probarlo, se debe hacer ***build*** del proyecto y se debe ***ejecutar*** (mantenerlo en ejecución). Se podrá verificar que el servicio se inició correctamente si ingresa a ***http://localhost:8080/ws/users.wsdl*** mediante el navegador.

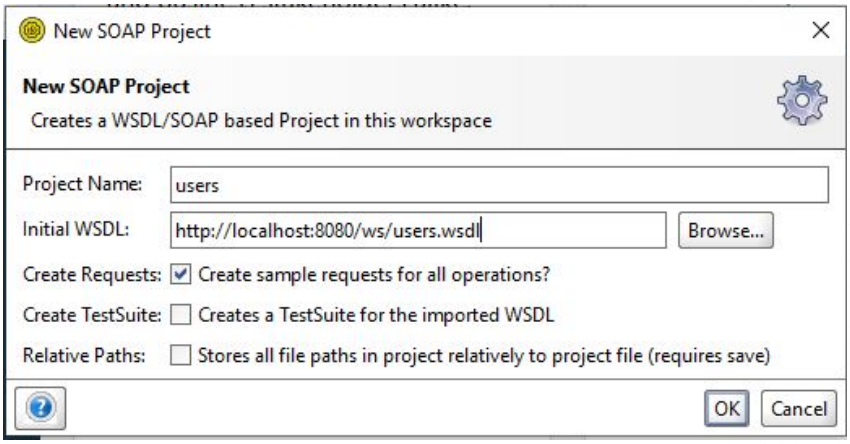


```
INFO 9544 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring MessageDispatcherServlet 'messageDispatcherServlet'
INFO 9544 --- [nio-8080-exec-1] o.s.w.t.http.MessageDispatcherServlet    : Initializing Servlet 'messageDispatcherServlet'
INFO 9544 --- [nio-8080-exec-1] o.s.ws.soap.saaj.SaajSoapMessageFactory  : Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
INFO 9544 --- [nio-8080-exec-1] o.s.w.t.http.MessageDispatcherServlet    : Completed initialization in 39 ms
```

14. Se puede verificar el estado de la tabla ***users***, tras la ejecución de cada una de las peticiones que se realizarán e igualmente la terminal, finalmente, abrir ***SoapUI***.



15. Crear un nuevo proyecto SOAP, agregar un nombre y la dirección del WSDL, http://localhost:8080/ws/users.wsdl. Seleccione la opción de crear peticiones de ejemplo para todas las operaciones.



16. Para **crear** un nuevo usuario, configurar la petición de la siguiente manera:

- Hacer clic en el botón "+" al lado izquierdo ***addUser***.
- Hacer doble clic sobre *Request 1*, desplegable de addUser.

- Request URL (en la entrada de texto de la parte superior): http://localhost:8080/ws
- Agregar los datos que desee al usuario en el cuerpo de la petición, como por ejemplo:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:user="http://localhost/users">
   <soapenv:Header/>
   <soapenv:Body>
      <user:addUserRequest>
         <user:firstname>Pepito</user:firstname>
         <user:lastname>Perez</user:lastname>
         <user:address>Cr 123 # 52 - 15</user:address>
         <user:genre>Masculino</user:genre>
         <user:age>35</user:age>
      </user:addUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

17. Hacer clic en el botón “Submit” para hacer el ***request***, a lo cual recibirá la siguiente respuesta:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:addUserResponse xmlns:ns2="http://localhost/users">
         <ns2:serviceStatus>
            <ns2:statusCode>SUCCESS</ns2:statusCode>
            <ns2:message>User adding succesfully</ns2:message>
         </ns2:serviceStatus>
         <ns2:user>
            <ns2:userId>3</ns2:userId>
            <ns2:firstname>Pepito</ns2:firstname>
            <ns2:lastname>Perez</ns2:lastname>
            <ns2:address>Cr 123 # 52 - 15</ns2:address>
            <ns2:genre>Masculino</ns2:genre>
            <ns2:age>35</ns2:age>
         </ns2:user>
      </ns2:addUserResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

18. Para **obtener** todos los usuarios, configurar la petición de la siguiente manera:

- Hacer clic en el botón “+” al lado izquierdo ***getAllUsers***.
- Hacer doble clic sobre *Request 1*, desplegable de getAllUsers.
- Request URL (en la entrada de texto de la parte superior): http://localhost:8080/ws

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:getAllUsersResponse xmlns:ns2="http://localhost/users">
         <ns2:user>
            <ns2:userId>2</ns2:userId>
            <ns2:firstname>Juan</ns2:firstname>
            <ns2:lastname>Sanchez</ns2:lastname>
            <ns2:address>Cr 13 # 12 - 05</ns2:address>
            <ns2:genre>Masculino</ns2:genre>
            <ns2:age>24</ns2:age>
         </ns2:user>
         <ns2:user>
            <ns2:userId>3</ns2:userId>
            <ns2:firstname>Pepito</ns2:firstname>
            <ns2:lastname>Perez</ns2:lastname>
            <ns2:address>Cr 123 # 52 - 15</ns2:address>
            <ns2:genre>Masculino</ns2:genre>
            <ns2:age>35</ns2:age>
         </ns2:user>
      </ns2:getAllUsersResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

19. Para **obtener** un usuario mediante su **Id**, configurar la petición de la siguiente manera:

- Hacer clic en el botón “+” al lado izquierdo ***getUserById***.
- Hacer doble clic sobre *Request 1*, desplegable de getUserById.
- Request URL (en la entrada de texto de la parte superior): http://localhost:8080/ws
- Agregar el id del usuario que se desee obtener, en el cuerpo de la petición, como por ejemplo:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:user="http://localhost/users">
   <soapenv:Header/>
   <soapenv:Body>
      <user:getUserByIdRequest>
         <user:userId>2</user:userId>
      </user:getUserByIdRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

20. Para **modificar** un usuario, configurar la petición de la siguiente manera:

- Hacer clic en el botón "+" al lado izquierdo ***updateUser***.
- Hacer doble clic sobre *Request 1*, desplegable de updateUser.
- Request URL (en la entrada de texto de la parte superior): http://localhost:8080/ws
- Agregar los datos que se desee, en el cuerpo de la petición, como por ejemplo:

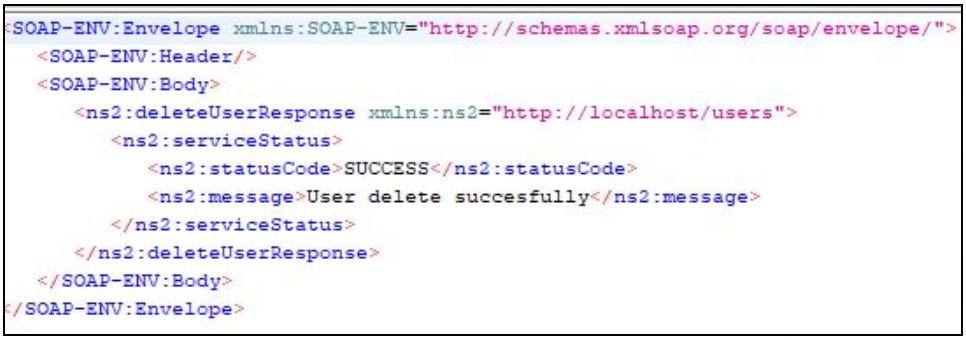
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:user="http://localhost/users">
   <soapenv:Header/>
   <soapenv:Body>
      <user:updateUserRequest>
         <user:userId>3</user:userId>
         <user:firstname>Pepito</user:firstname>
         <user:lastname>Perez</user:lastname>
         <user:address>Cr 15 # 12 - 15</user:address>
         <user:genre>Masculino</user:genre>
         <user:age>36</user:age>
      </user:updateUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:updateUserResponse xmlns:ns2="http://localhost/users">
         <ns2:serviceStatus>
            <ns2:statusCode>SUCCESS</ns2:statusCode>
            <ns2:message>User update succesfully</ns2:message>
         </ns2:serviceStatus>
      </ns2:updateUserResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

21. Para **eliminar** un usuario mediante su **Id**, configurar la petición de la siguiente manera:

- Hacer clic en el botón “+” al lado izquierdo ***deleteUser***.
- Hacer doble clic sobre *Request 1*, desplegable de deleteUser.
- Request URL (en la entrada de texto de la parte superior): http://localhost:8080/ws
- Agregar el id del usuario que se desea eliminar, en el cuerpo de la petición, como por ejemplo:

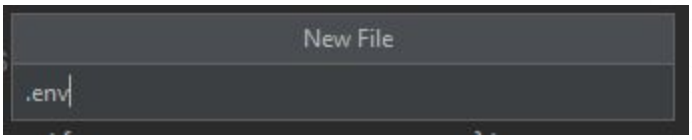
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:user="http://localhost/users">
   <soapenv:Header/>
   <soapenv:Body>
      <user:deleteUserRequest>
         <user:userId>3</user:userId>
      </user:deleteUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```



```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:deleteUserResponse xmlns:ns2="http://localhost/users">
         <ns2:serviceStatus>
            <ns2:statusCode>SUCCESS</ns2:statusCode>
            <ns2:message>User delete succesfully</ns2:message>
         </ns2:serviceStatus>
      </ns2:deleteUserResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

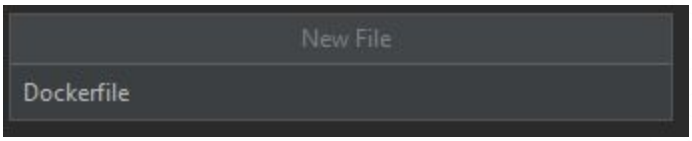
Despliegue

1. Para ejecutar el ejercicio mediante ***Docker***, se deberá primero incorporar las variables de entorno, para lo cual se debe crear un archivo ***.env*** en la raíz del proyecto. Agregar el siguiente código:



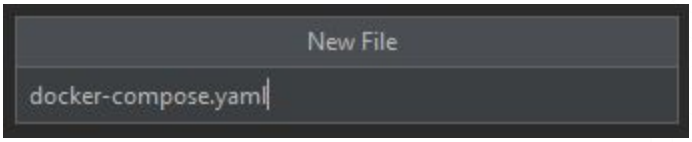

```
USERS_DB_PASSWORD=postgres
USERS_DB_DNS=users_db
USERS_DB_USER=postgres
USERS_DB_NAME=users_db
PGDATA=/var/lib/postgresql/data/pgdata
```

2. Crear un archivo ***Dockerfile*** en la raíz del proyecto y agregar el siguiente código[link del archivo]:




```
FROM        maven:3.6.3-openjdk-8 as compiler
COPY        ./ /root/users
WORKDIR     /root/users/
RUN         mvn clean package -DskipTests

FROM        openjdk:8-jdk
RUN         mkdir /root/app
COPY                                                --from=compiler
/root/users/target/users-0.0.1-SNAPSHOT.jar
/root/app/users-0.0.1-SNAPSHOT.jar
ENTRYPOINT  ["java", "-jar", "/root/app/users-0.0.1-SNAPSHOT.jar"]
EXPOSE      8080
```

3. Crear un archivo ***docker-compose.yaml*** en la raíz del proyecto y agregar el siguiente código:



```yaml
version: "3"

services:
 user_cp:
   container_name: user_cp
   build: .
   restart: always
   ports:
     - 8080:8080
   environment:
     - USERS_DB_PASSWORD
     - USERS_DB_DNS
     - USERS_DB_USER
     - USERS_DB_NAME
   depends_on:
     - users_db

 users_db:
   container_name: users_db
   image: postgres
   restart: always
   ports:
     - 5432:5432
   environment:
     - PGDATA=/var/lib/postgresql/data/pgdata
     - POSTGRES_PASSWORD=${USERS_DB_PASSWORD}
     - POSTGRES_USER=${USERS_DB_USER}
     - POSTGRES_PORT=5432
     - POSTGRES_DB=${USERS_DB_NAME}
```

4. En la terminal, desde la raíz del proyecto, ejecutar los siguientes comandos:

   - **Creación de imagen Docker:**

     ```
     sudo docker-compose build
     ```

```
Removing intermediate container 18170a94db7d
 ---> b9aa46d3d2b4
Step 5/9 : FROM        openjdk:8-jdk
 ---> 192ceee8f2fd
Step 6/9 : RUN         mkdir /root/app
 ---> Using cache
 ---> 0740d087dc57
Step 7/9 : COPY        --from=compiler /root/users/target/users-0.0.1-SNAPSHOT.jar /root/app/users-0.0.1-SNAPSHOT.jar
 ---> a1b97ff85083
Step 8/9 : ENTRYPOINT  ["java", "-jar", "/root/app/users-0.0.1-SNAPSHOT.jar"]
 ---> Running in b96fb030f16f
Removing intermediate container b96fb030f16f
 ---> b13222b0107e
Step 9/9 : EXPOSE      8080
 ---> Running in 281aca926e23
Removing intermediate container 281aca926e23
 ---> 1f287d3ab502
Successfully built 1f287d3ab502
Successfully tagged users_user_cp:latest
```

- **Creación y despliegue de contenedores:**

  ```
  sudo docker-compose up
  ```

```
users_db    |
users_db    | PostgreSQL init process complete; ready for start up.
users_db    |
users_db    |                       UTC [1] LOG:  starting PostgreSQL 13.1 (Debian 13.1-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
users_db    |                       UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
users_db    |                       UTC [1] LOG:  listening on IPv6 address "::", port 5432
users_db    |                       UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
users_db    |                       UTC [66] LOG:  database system was shut down at
users_db    |                       UTC [1] LOG:  database system is ready to accept connections
user_cp     |
user_cp     |   .   ____          _            __ _ _
user_cp     |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
user_cp     | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
user_cp     |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
user_cp     |   '  |____| .__|_| |_|_| |_\__, | / / / /
user_cp     |  =========|_|==============|___/=/_/_/_/
user_cp     |  :: Spring Boot ::                (v2.4.0)
user_cp     |
user_cp     |                       INFO 1 --- [           main] com.user_cp.users.UsersApplication       : Starting UsersApplication v0.0.1-SNAPSHOT using Java 1.8.0_275 on e8db8bcdf6c7 with PID 1 (/root/app
/users-0.0.1-SNAPSHOT.jar started by root in /)
user_cp     |                       INFO 1 --- [           main] com.user_cp.users.UsersApplication       : No active profile set, falling back to default profiles: default
user_cp     |                       INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
user_cp     |                       INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 133 ms. Found 1 JPA repository interfaces.
user_cp     |                       INFO 1 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'soapWSConfig' of type [com.user_cp.users.resource.soap.SoapWSConfig$$EnhancerBySpringCGLIB$$1a
849472] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
user_cp     |                       INFO 1 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotation.DelegatingWsConfiguration' of type [org.springframewo
rk.ws.config.annotation.DelegatingWsConfiguration$$EnhancerBySpringCGLIB$$46ab34a0] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
user_cp     |                       INFO 1 --- [           main] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
user_cp     |                       INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
user_cp     |                       INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
user_cp     |                       INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.39]
user_cp     |                       INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
user_cp     |                       INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 3927 ms
user_cp     |                       INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
user_cp     |                       INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
user_cp     |                       INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.23.Final
user_cp     |                       INFO 1 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
user_cp     |                       INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
user_cp     |                       WARN 1 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during v
iew rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
user_cp     |                       INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
user_cp     |                       INFO 1 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
user_cp     |                       INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
user_cp     |                       INFO 1 --- [           main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories…
user_cp     | Hibernate: create table users (id  serial not null, address varchar(255), age int4, firstname varchar(255), genre varchar(255), lastname varchar(255), primary key (id))
user_cp     |                       INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal
.NoJtaPlatform]
user_cp     |                       INFO 1 --- [           task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
user_cp     |                       INFO 1 --- [           main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
user_cp     |                       INFO 1 --- [           main] com.user_cp.users.UsersApplication       : Started UsersApplication in 9.752 seconds (JVM running for 14.798)
```

5. Tras realizar la actividad inmediatamente anterior, quedarán desplegados dos contenedores en la máquina, uno para el endpoint de los servicios web (en el puerto 8080) y otro para la base de datos (en el puerto 5432). Para verificar, se debe ejecutar el siguiente comando en una pestaña de la terminal, diferente a la que se usó para el despliegue (paso anterior):

   ```
   docker ps
   ```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e8db8bcdf6c7	users_user_cp	"java -jar /root/app…"	2 minutes ago	Up About a minute	0.0.0.0:8080->8080/tcp	user_cp
4315e5753d57	postgres	"docker-entrypoint.s…"	2 minutes ago	Up 2 minutes	0.0.0.0:5432->5432/tcp	users_db

6. Finalmente, se podrán realizar todas las acciones ejecutadas anteriormente mediante **Postman** y **SoapUI**.