Examen 3_ ANA LAURA VÁZQUEZ SOLACHE

1. Escribe un escenario donde creaste una rama más y modificaste el mismo archivo. (Generar el merch y resolver el conflicto)

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (main)
$ git branch otra-rama

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (main)
$ git branch
* main
  otra-rama

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (main)
$ git switch otra-rama
Switched to branch 'otra-rama'

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (otra-rama)
$ git branch
  main
* otra-rama
```

Para llevarlo a cabo a través de la líneas de comando, necesitamos abrir la terminal en el directorio en el cual tenemos el repositorio de git, en este podemos crear otra rama con el comando - git branch “nombre de la rama”- siguiendo el ejemplo de la imagen podemos llevarlo a cabo de la siguiente manera - **git branch otra-rama**-, nos cambiamos a esta rama a través del comando -**git switch otra-rama-** (es importante poner el nombre de la rama a la que nos queremos cambiar, si sabemos las letras con las que comienza el nombre de la rama, podemos presionar el tabulador y este nos autocompleta si la encontró e identifica). O para ahorrarnos el cambio de rama podemos utilizar el comando de -**git switch -c cambioSw o -git checkout -b ramaCambio-**

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (otra-rama)
$ git checkout -b ramaCambio
Switched to a new branch 'ramaCambio'

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (ramaCambio)
$ git branch
  main
  otra-rama
* ramaCambio
```

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (ramaCambio)
$ git switch -c cambioSw
Switched to a new branch 'cambioSw'

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git branch
* cambioSw
  main
  otra-rama
  ramaCambio
```

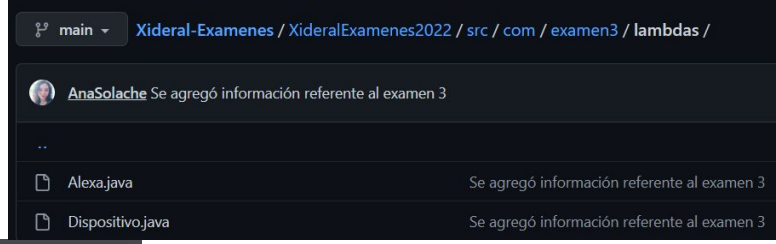
Una vez se llevan a cabo modificaciones de los archivos, podemos ver cuáles fueron estas modificaciones a través de un **-git status-**, este nos indica que documentos nos falta subir y podemos elegir uno a uno que documento subir a través de un **git add "nombre del archivo"** o un **- git add . -** con esto añadiremos todas las modificaciones, teniendo cuidado de que estamos en la rama que queremos modificar. Hacemos un commit a través del comando **-git commit -m "Comentario descriptivo de la modificación"-**, describiendo claramente cuáles fueron nuestras modificaciones, y finalmente lo enviamos al repositorio a través de un **-git push origin develop-** (o el nombre de la rama a la que quieres enviar el cambio), cuando queramos fusionar ramas podemos enviar el comando de **-git merge origin main-,** una vez hecho esto solo dice Already up to date, en caso de que se haya modificado dentro del mismo archivo, saldrá un conflicto, podemos ver las diferencias entre las ramas y el documento utilizando el siguiente comando **"git diff main",** y para resolver el **conflicto** simplemente abrimos el archivo en el editor de código y las modificaciones van a aparecer entre **<<<<< -------- >>>>>>** hacemos las modificaciones dentro del código requerido, o igual en el caso de visual studio, aparecen las opciones de **accept incoming change, accept both changes, o accept current change** y lo único que se tiene que hacer es elegir la opción que corresponda e intentar de nuevo el merge sin condlictos.

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git status
On branch cambioSw
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   OtroHolaMundo.txt

no changes added to commit (use "git add" and/or "git commit -a")

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git add OtroHolaMundo.txt

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git commit -m"Se agregó otro documento para ver el status y merge sin conflictos"
[cambioSw 881e9af] Se agregó otro documento para ver el status y merge sin conflictos
 1 file changed, 1 insertion(+), 1 deletion(-)

anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git push origin cambioSw
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 363 bytes | 121.00 KiB/s, done.
```

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (cambioSw)
$ git merge main
Already up to date.
```

```
anaso@SOL-G5 MINGW64 ~/Desktop/PruebaGit (otra-rama)
$ git diff main
diff --git a/HolaMundo.txt b/HolaMundo.txt
index 0150e81..3fe2932 100644
--- a/HolaMundo.txt
+++ b/HolaMundo.txt
@@ -1 +1 @@
-Este es un Hola Mundo desde main
\ No newline at end of file
+Este es un Hola Mundo desde otra-rama
\ No newline at end of file
```

2. Realiza el ejercicio de tu interfaz del primer examen con polimorfismo y lambas.



url attached to the image

Alexa.java

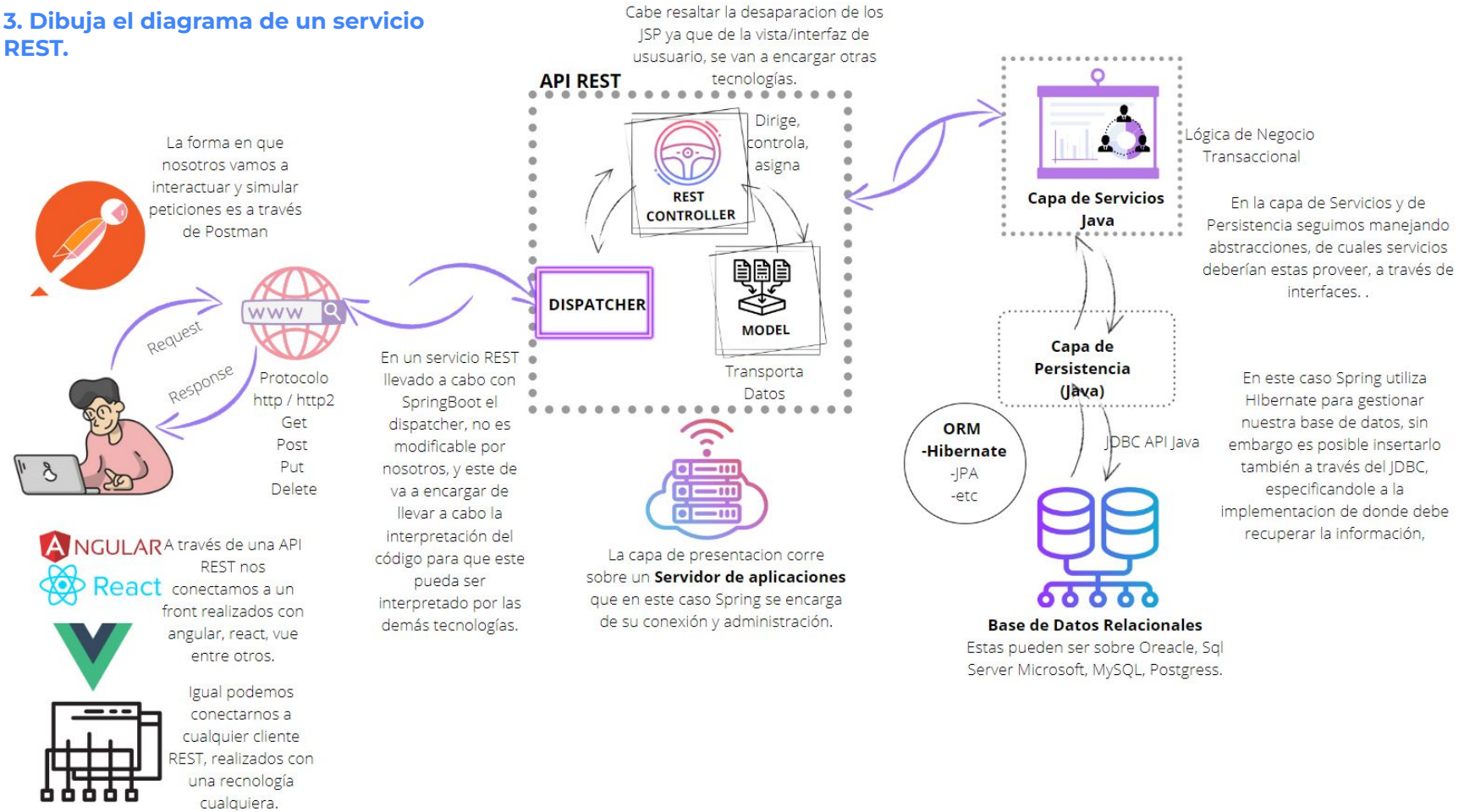
```java
package com.examen3.lambdas;

import java.util.*;

public class Alexa {

    public static void main(String[] args) {

        //Se creo ArrayList con dispositivos
        List <Dispositivo> listaDispositivos =  new ArrayList<>();
        listaDispositivos.add(new Dispositivo("Leonardo","Telefono","Huawei",2));
        listaDispositivos.add(new Dispositivo("Familia","Television","Samsung",5));
        listaDispositivos.add(new Dispositivo("Leonardo","SmarthWach","Huawei",1));
        listaDispositivos.add(new Dispositivo("Angela","Celular","Samsung",2));
        listaDispositivos.add(new Dispositivo("Angela","SmarthWach","Samsung",1));
        listaDispositivos.add(new Dispositivo("Ana","Celular","Motorola",1));

        //Lambdas Consumer retornan métodos void
        Consumer<Dispositivo> prender = dis -> System.out.println("Alexa prendio "+dis.getUso()+" de " + dis.getPropietario());
        Consumer<Dispositivo> conectar = dis -> System.out.println("Alexa se conecto "+dis.getUso()+" de " + dis.getPropietario());
        Consumer<Dispositivo> apagar = dis -> System.out.println("Alexa apago "+dis.getUso()+" de " + dis.getPropietario());

        //Impresion y uso de lambdas Consumer, por cada elemento del ArrayList, con indicaciones
        System.out.println("********************");
        System.out.println("Se obtienen 3 lambdas Consumer con forEach de todos los elementos del ArrayList = ");
        System.out.println("********************");
        listaDispositivos.forEach(prender);
        System.out.println("********************");
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Terminal | Console | JUnit

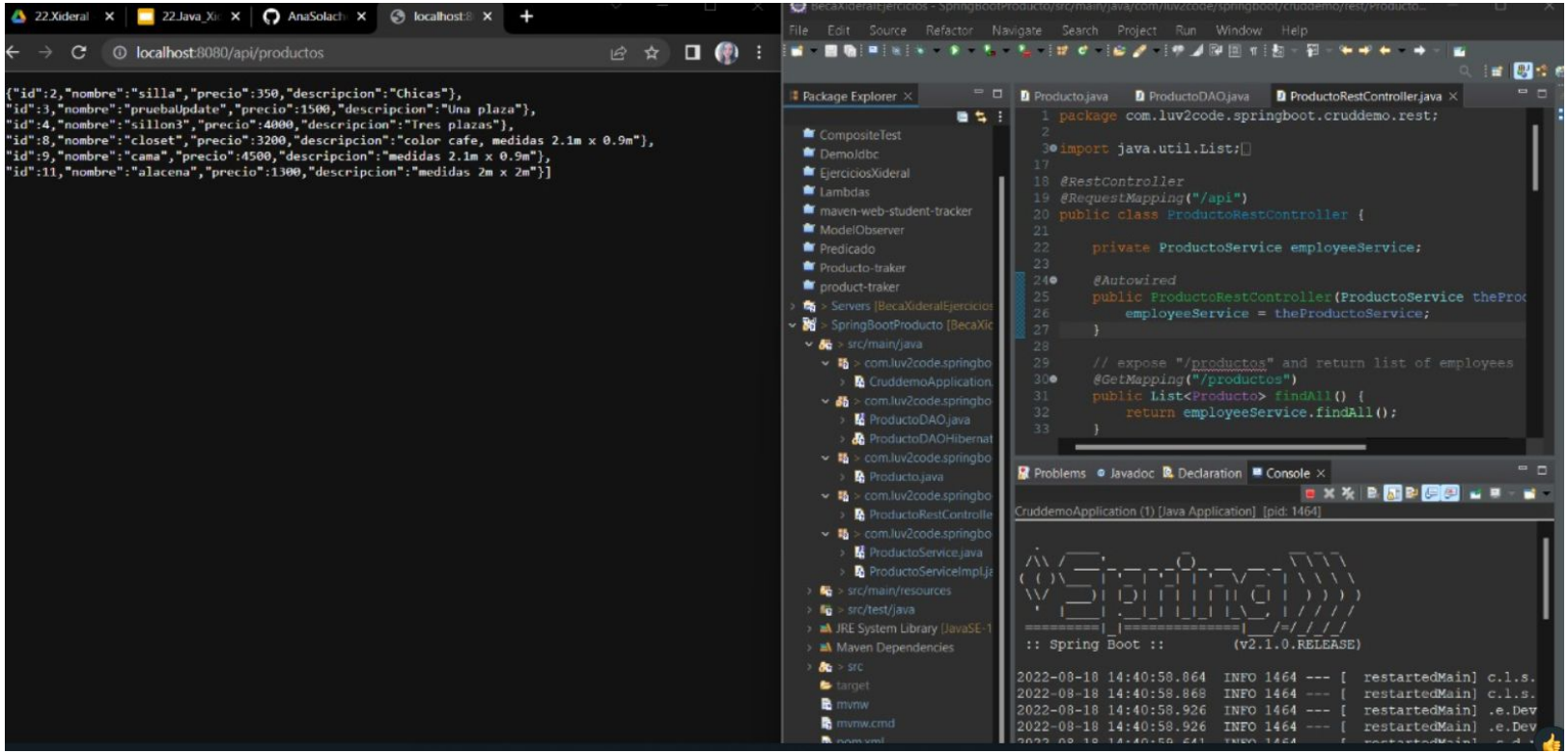
<terminated> Alexa [Java Application] C:\Users\anaso\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_11.0.16.v20220805-0923\jre\bin\javaw.exe (19 ago 2022 12:04:14 – 12:04:15) [pid

```
********************
Se obtienen 3 lambdas Consumer con forEach de todos los elementos del ArrayList =
********************
Alexa prendio Telefono de Leonardo
Alexa prendio Television de Familia
Alexa prendio SmarthWach de Leonardo
Alexa prendio Celular de Angela
Alexa prendio SmarthWach de Angela
Alexa prendio Celular de Ana
********************
Alexa se conecto Telefono de Leonardo
```

3. Dibuja el diagrama de un servicio REST.



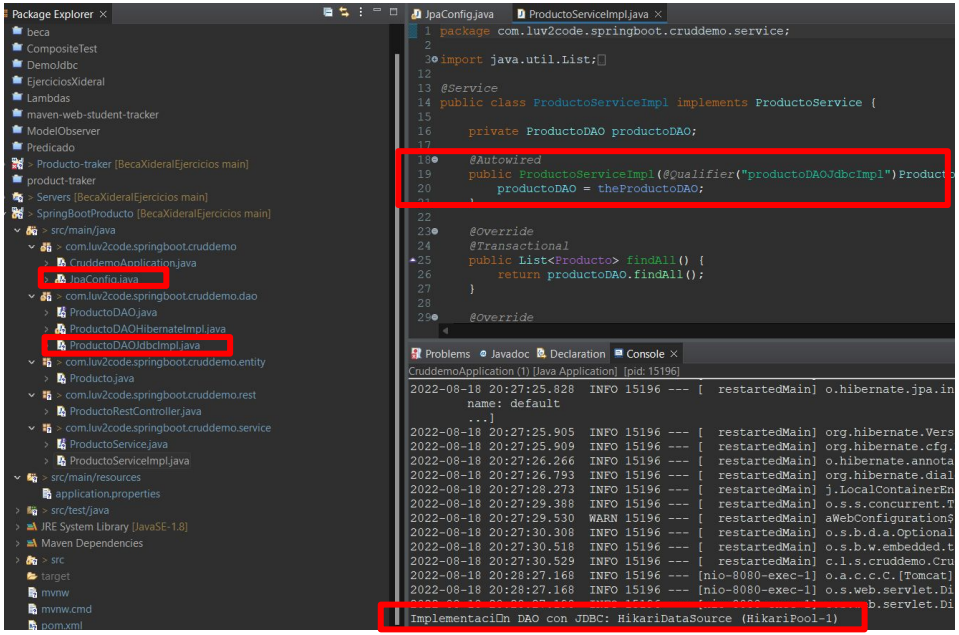
4. Implementar tu servicio REST personalizado.

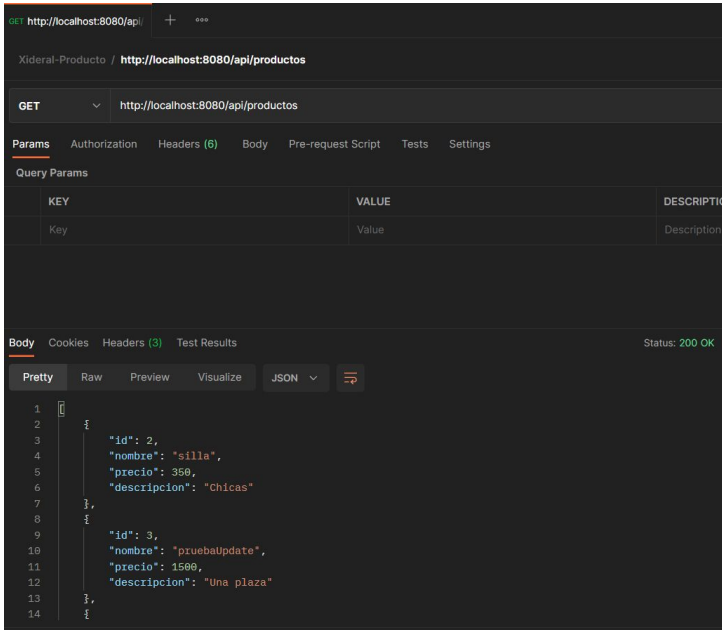


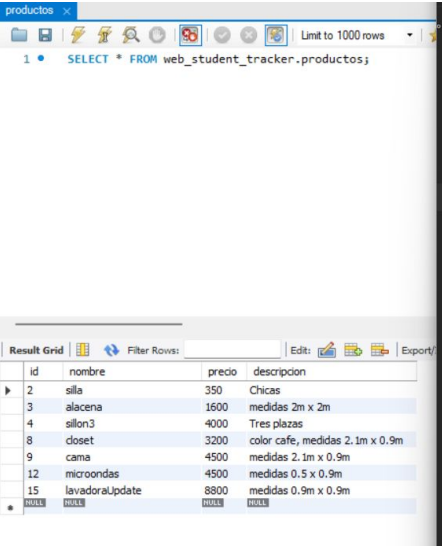
url attached to the image

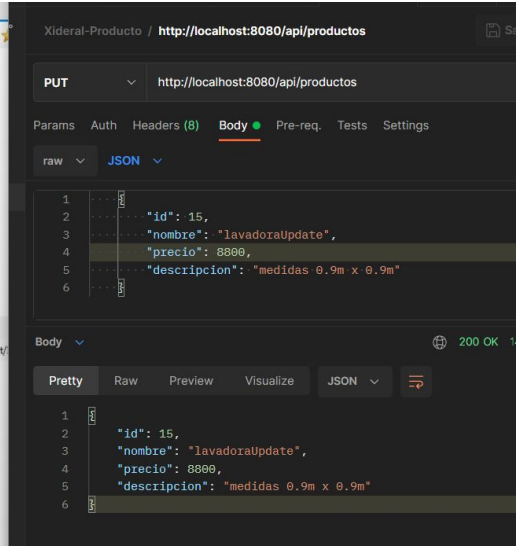
Como se comentó en el diagrama, **SpringBoot utiliza Hibernate** como filtro entre la capa de persistencia y las bases de datos, se encarga de “traducir” la información que obtenemos de la base de datos, en este caso para utilizar también una **JDBC que es el API de Java** para conexión con nuestra base de datos insertamos una nueva clase en la cual designamos la configuración de conexión de nuestra **JDBC (JpaConfig)** y a través de su implementación (Data Acces Object “DAOImpl”) designamos los métodos de load, getId, add & delete.

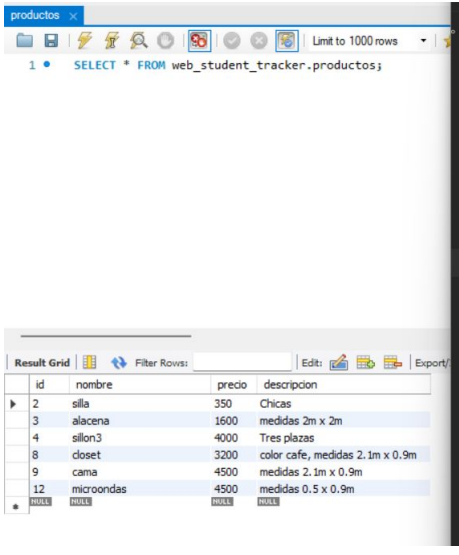
Y es a través de **la capa de servicio** que es el que obtiene a nuestro DAO, en su implementación a través del @Qualifier (“nombreQueIdenficicaDAOaUsar”) en este caso **@Qualifier (“productoJbdcImpl)** que es el nombre que designamos en nuestra configuración para nombrar el recurso.
Finalmente simulamos en Postman la petición GET, obteniendo también en consola la impresión de donde fue obtenida esta información como especificamos en código.

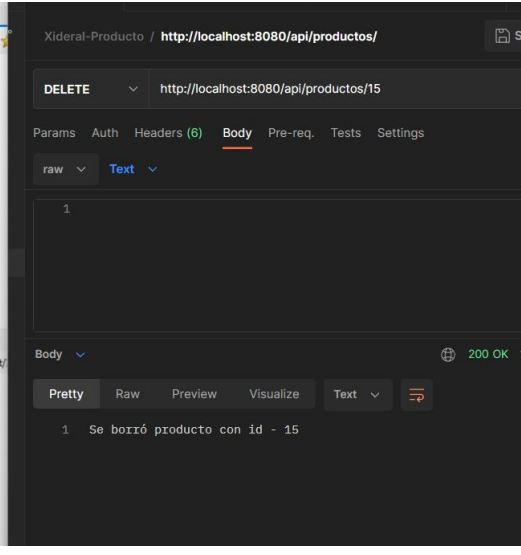


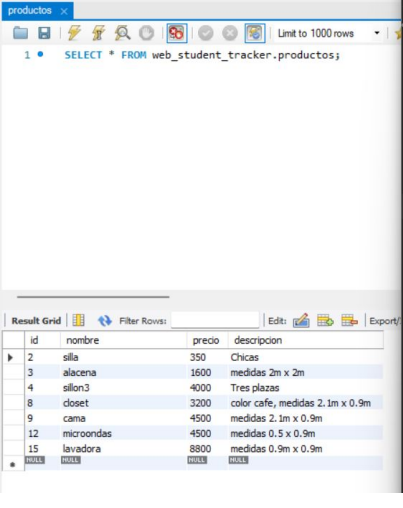


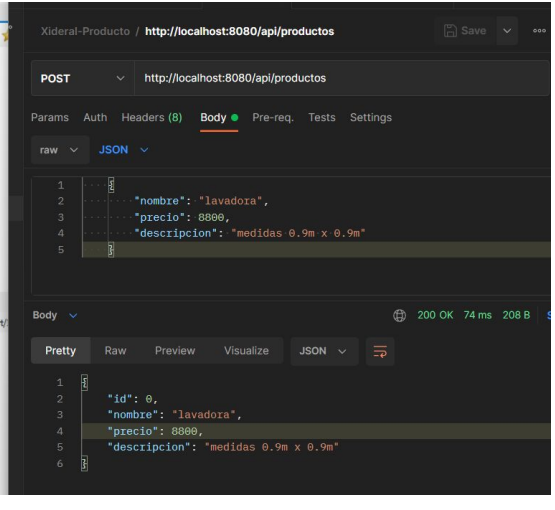












Validación de servicios REST con Postman (POST, PUT, DELETE).

5. Explica que es inyección de dependencias

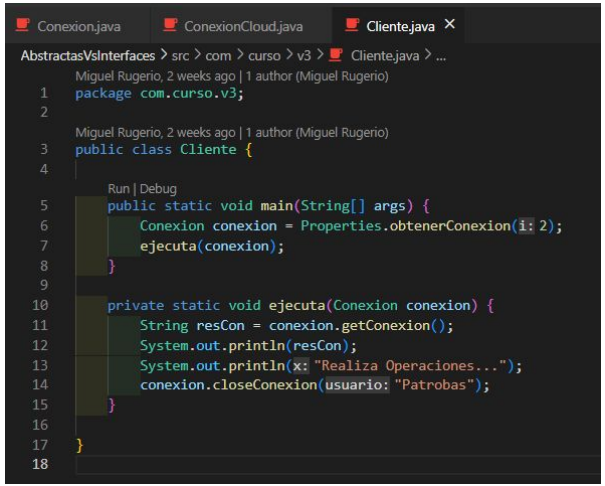
La inyección de dependencias es un patrón de diseño que se encarga de delegar responsabilidades de creación de instancias, esto para generar una alta cohesión con un bajo acoplamiento. Ejemplos de esto puede ser el patrón de diseño “observer”, en el cual generamos interfaces las cuales inyectan características a nuestras instancias (también se encuentra altamente ligado a los paradigmas de abstracción y de polimorfismo.)

Para evitar problemas en códigos por acoplamientos rígidos, este paradigma le proporciona sus dependencias en tiempo de ejecución por una entidad externa que coordina cada objeto en el sistema.

Los mecanismos que tenemos para inyectar es a través de inyección por variable, inyección a través de un Setter e inyección a través de un constructor.

En este ejemplo podemos ver como la clase cliente a través del método main obtiene una conexión la cual, a través de este código no sabemos cual es, ya que esta se va a resolver en tiempo de ejecución.

Se está inyectando la conexión, con una bajo acoplamiento al no tener que modificar a la clase, más que el número a través del cual obtendrá la conexión.



La conexión es una clase abstracta generando que las clases que extiendan de esta, deben de tener dos métodos en los cuales regresen un String y un Boolean respectivamente.

```java
// Conexion.java
// AbstractasVsInterfaces > src > com > curso > v3 > Conexion.java > ...
// Miguel Rugerio, 2 weeks ago | 1 author (Miguel Rugerio)
package com.curso.v3;

// Miguel Rugerio, 2 weeks ago | 1 author (Miguel Rugerio)
public abstract class Conexion {

    abstract String getConexion();

    abstract boolean closeConexion(String usuario);
}
```

Las conexiones que pueden ser a través de Cloud, Oracle, MySql definiendo sus comportamientos independendientes heredados a través de un @override

Todo esto con el fin de tener una alta cohesión con un bajo acoplamiento.

```java
// ConexionCloud.java
// AbstractasVsInterfaces > src > com > curso > v3 > ConexionCloud.java > { } com.curso.v3
// Miguel Rugerio, 2 weeks ago | 1 author (Miguel Rugerio)
package com.curso.v3;

import java.time.LocalDateTime;

// Miguel Rugerio, 2 weeks ago | 1 author (Miguel Rugerio)
public class ConexionCloud extends Conexion {

    @Override
    public String getConexion() {
        //Logica conexión
        return "Conexion exitosa Cloud";
    }

    @Override
    public boolean closeConexion(String usuario) {
        System.out.println("Guarda Logs "+usuario+
                " cerro conexion Cloud, "+LocalDateTime.now());
        //Lógica cerrar conexión
        return true;
    }

}
```