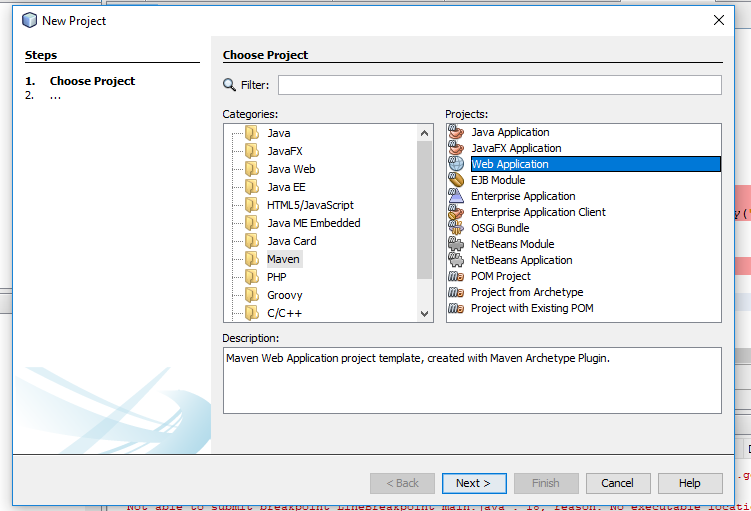
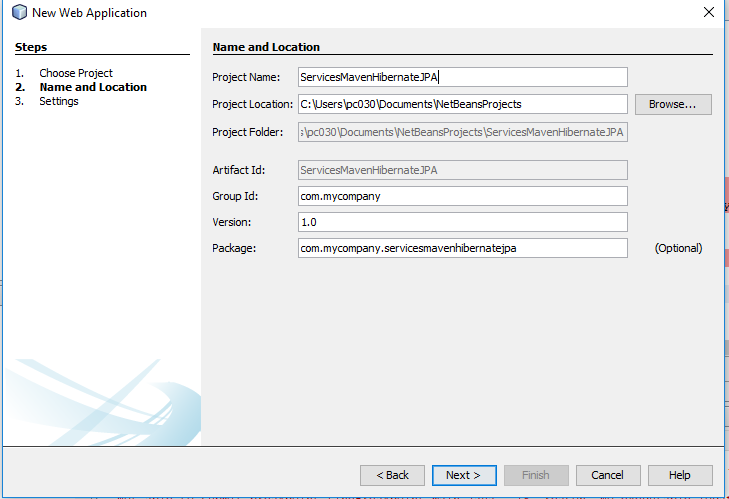
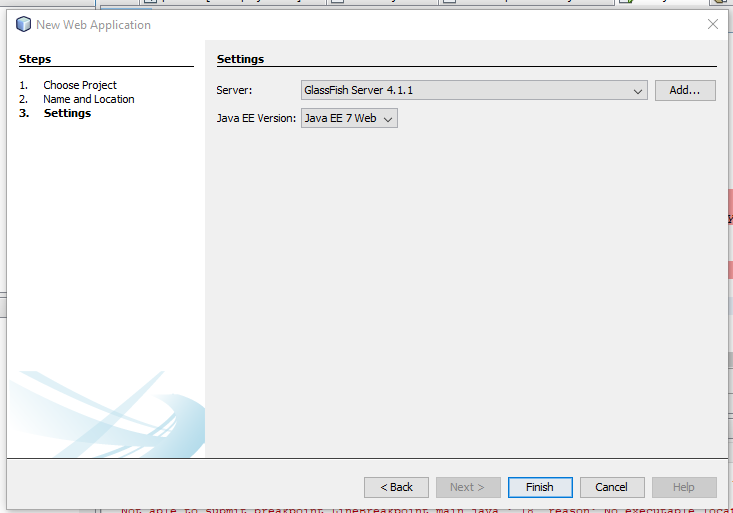
Proyecto JAVA MAVEN + JPA + HIBERNATE

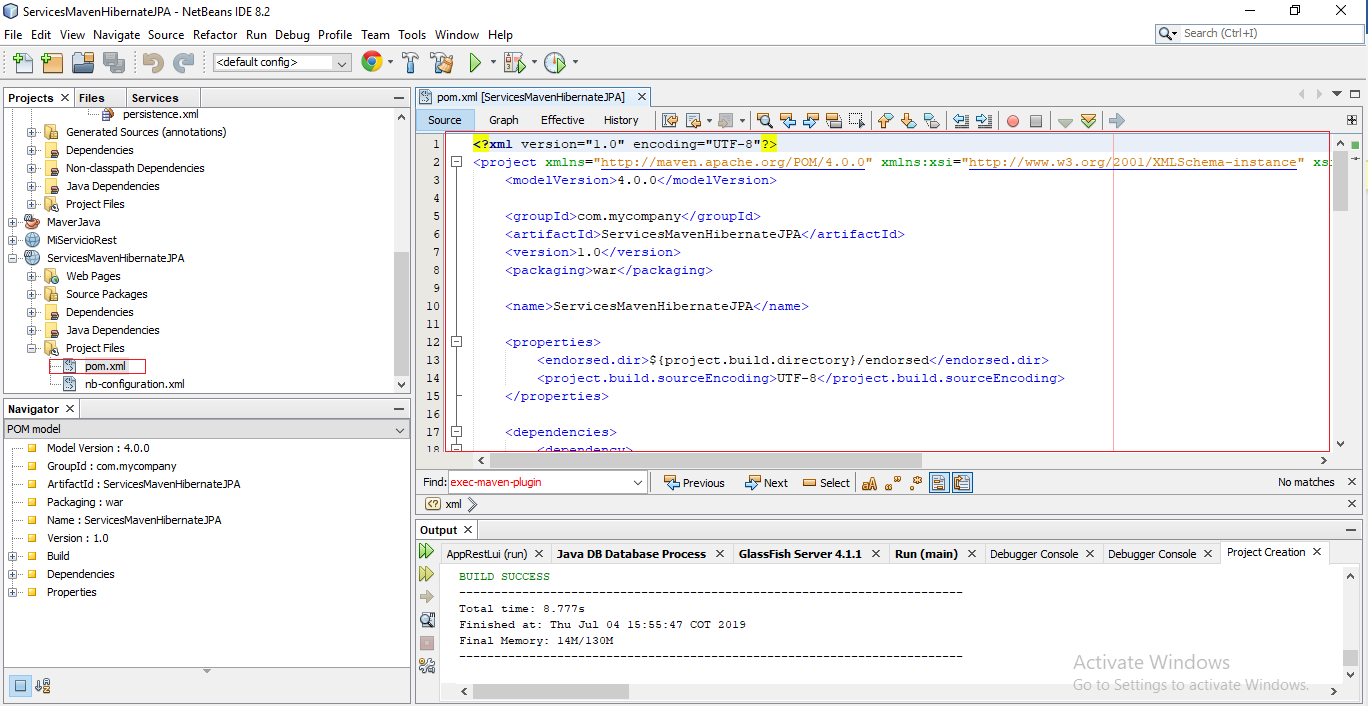
1. Crear un Proyecto Web Maven





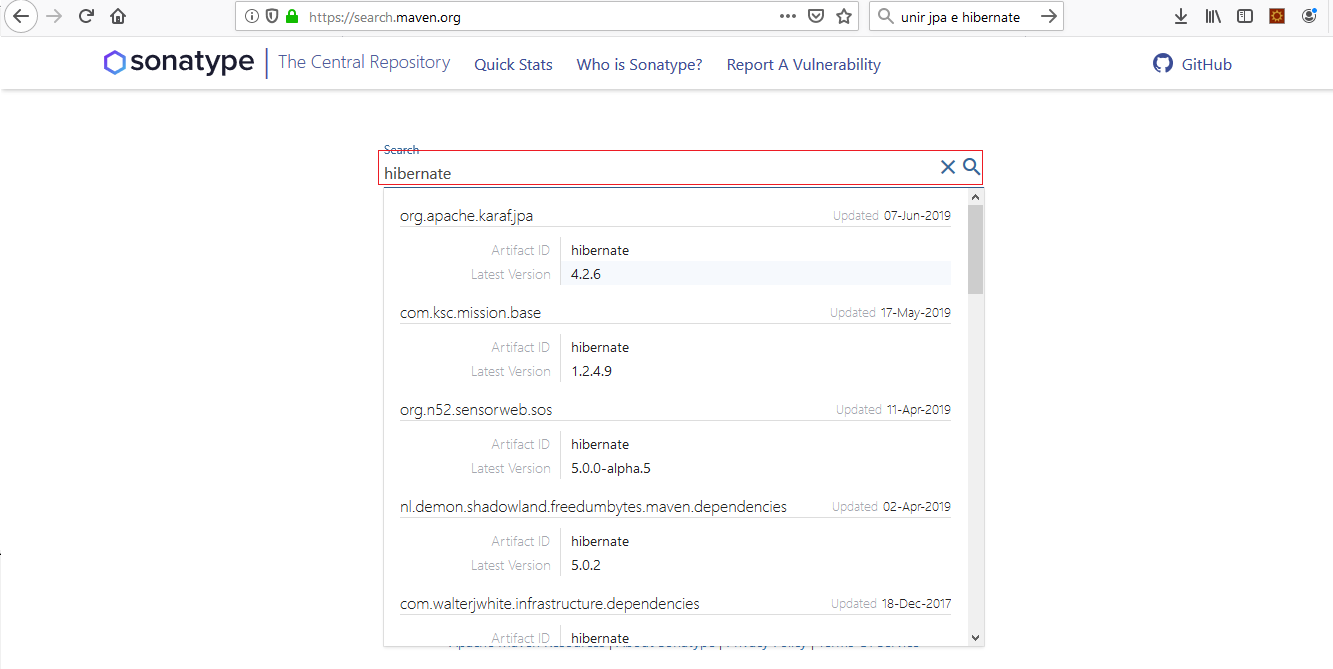


Se va a crear el archivo porm.xml es el archivo donde se deberan colocar las dependencias del proyecto estas se bajaran automaticamente en la compilación.

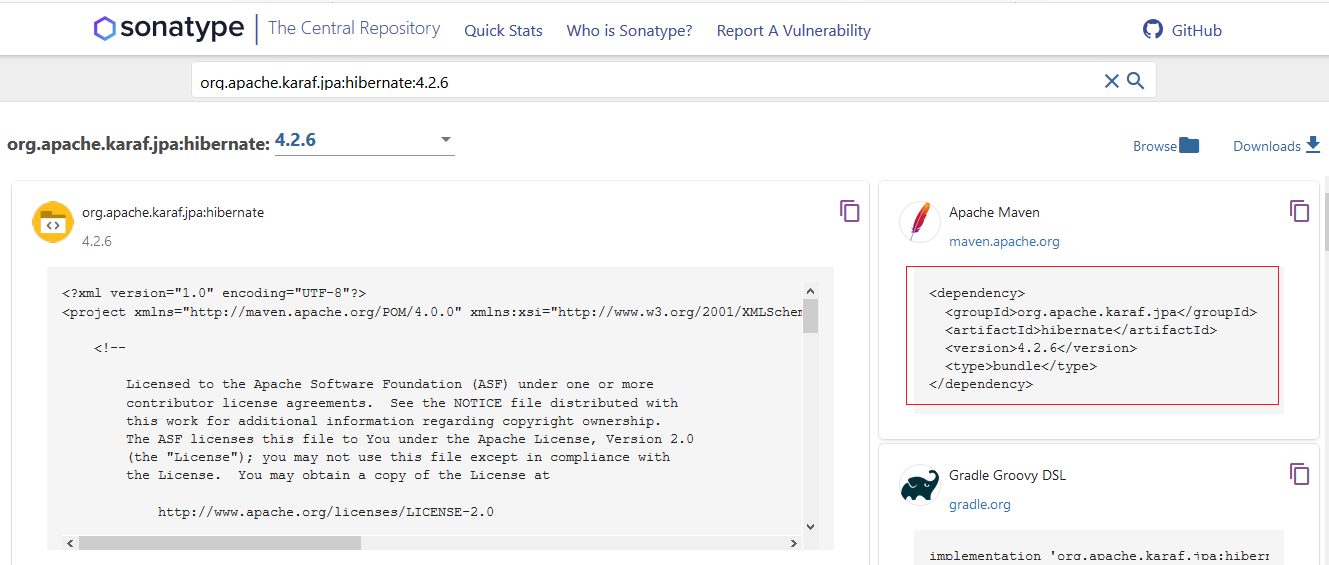


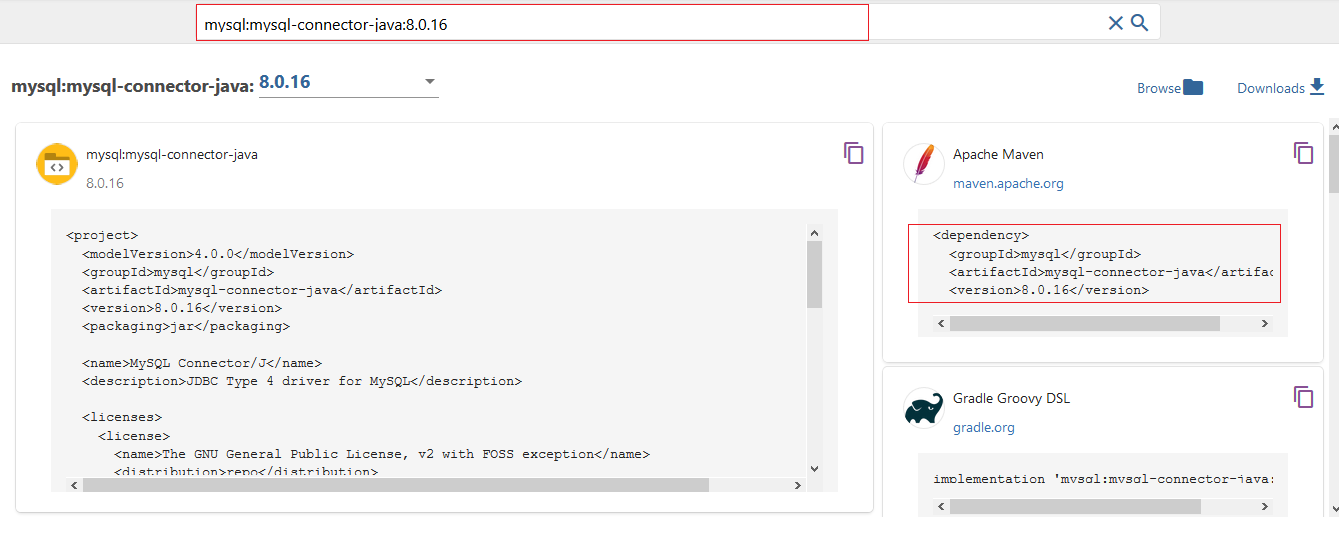
Vamos a buscar las librerias que necesitemos en el sito official de maven

<https://search.maven.org/>



Copiamos el xml de descarga y lo colocamos en nuestro porm.xml





<dependencies>

<dependency>

<groupId>org.eclipse.persistence</groupId>

<artifactId>eclipselink</artifactId>

<version>2.5.2</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.eclipse.persistence</groupId>

<artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>

<version>2.5.2</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-web-api</artifactId>

<version>7.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.16</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate</artifactId>

<version>3.6.0.Beta2</version>

<type>pom</type>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.6.Final</version>

</dependency>

<dependency>

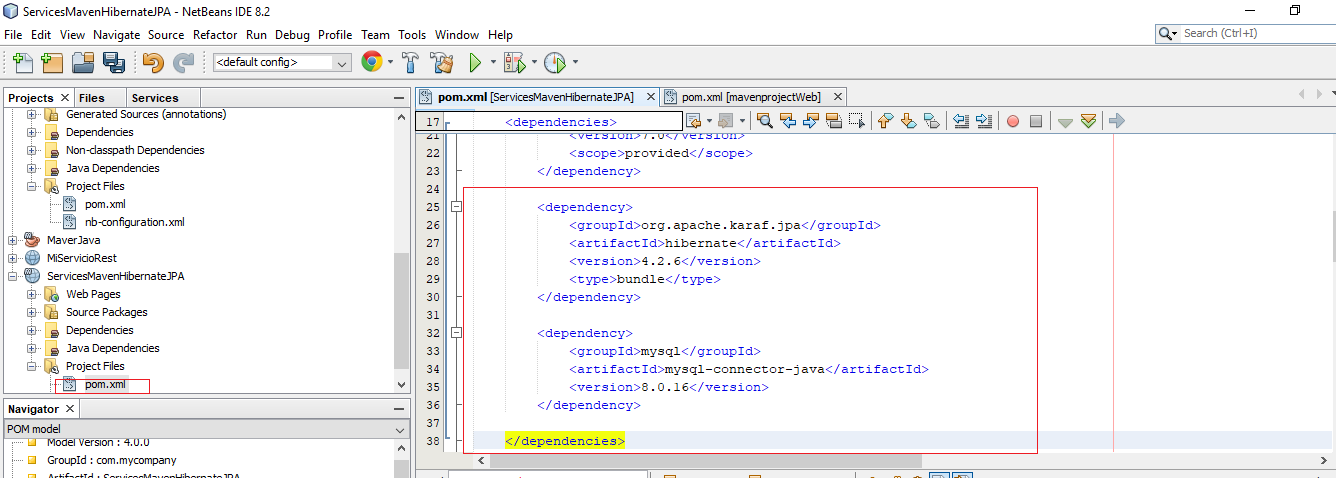
<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>4.3.6.Final</version>

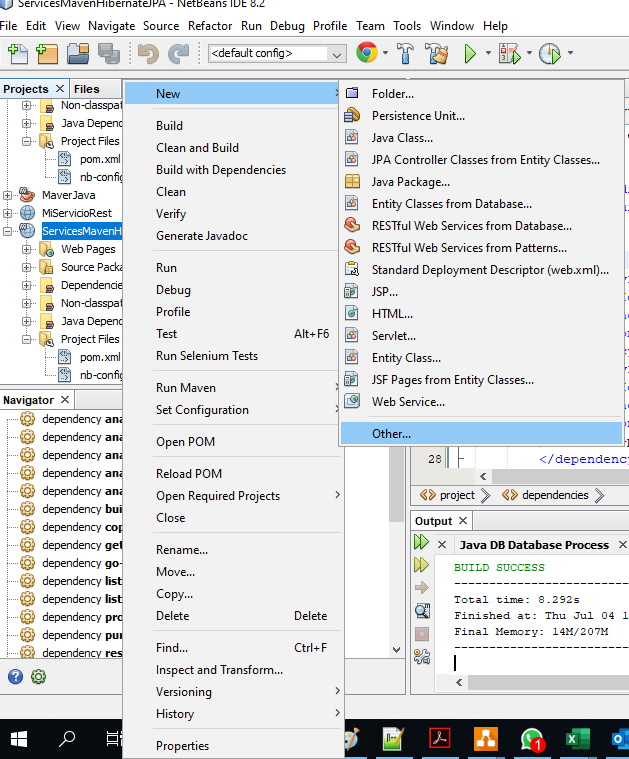
</dependency>

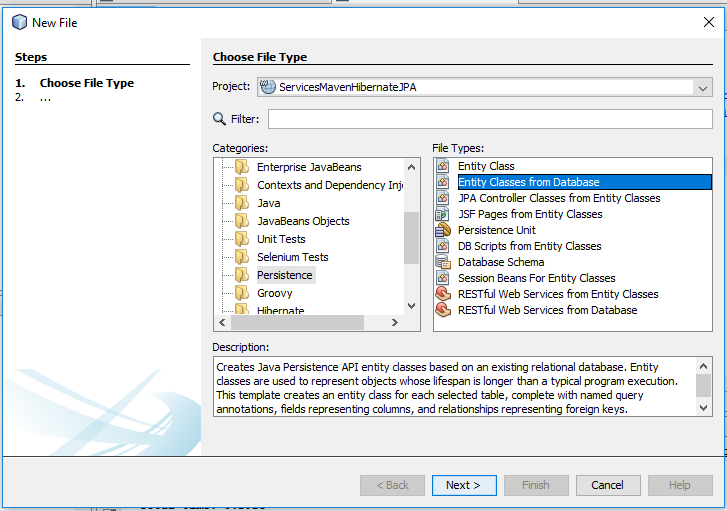
</dependencies>

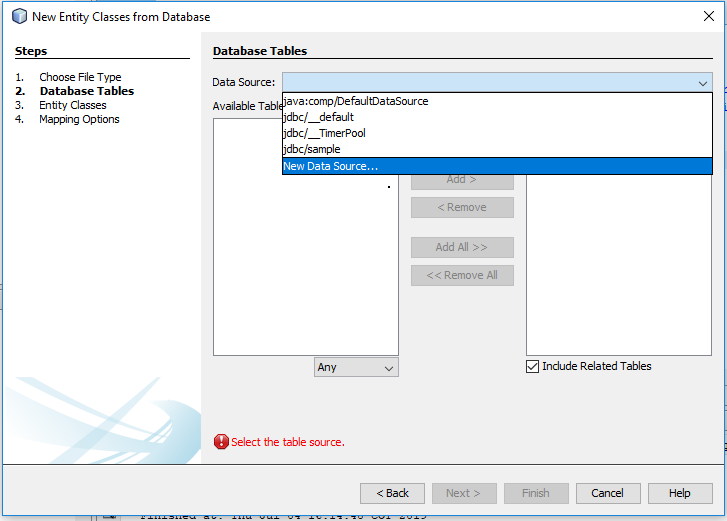


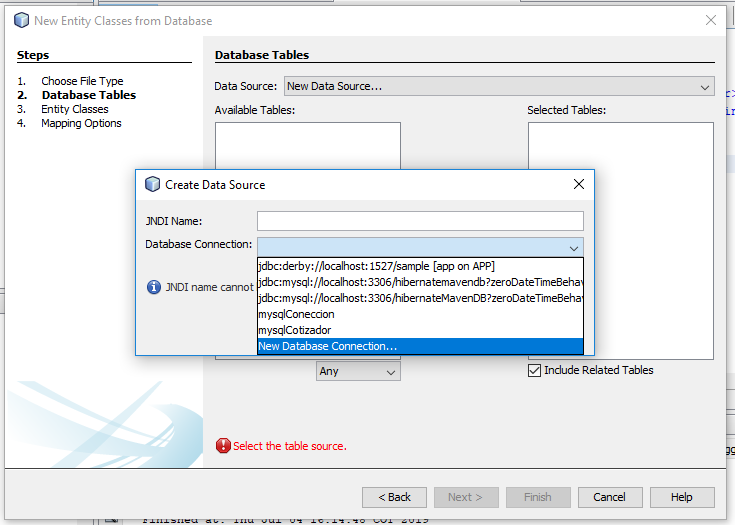
Lo Guardamos y compilamos

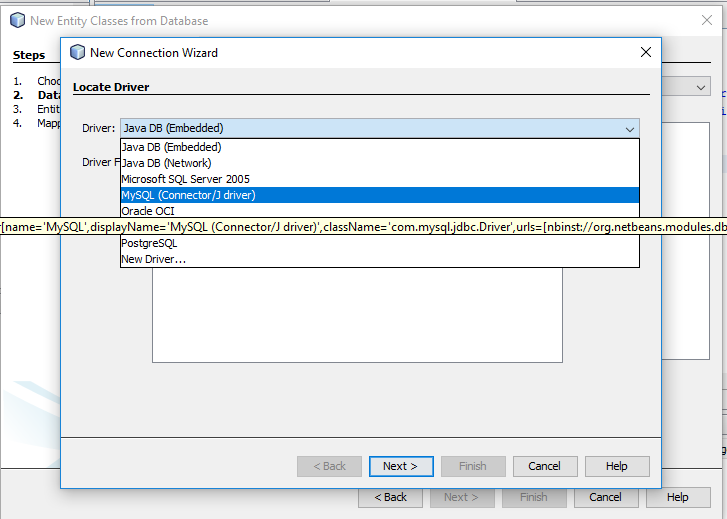
Agregar JPA

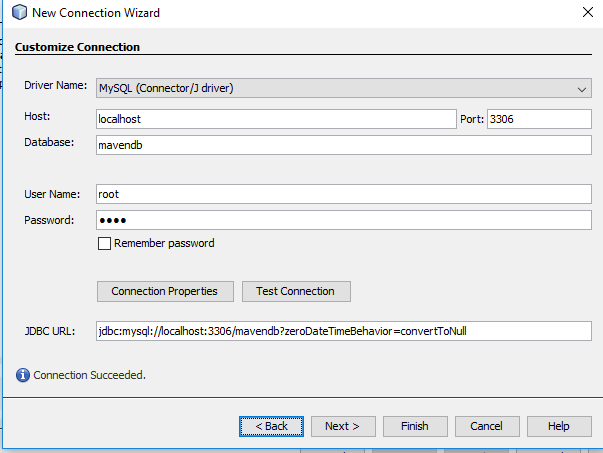


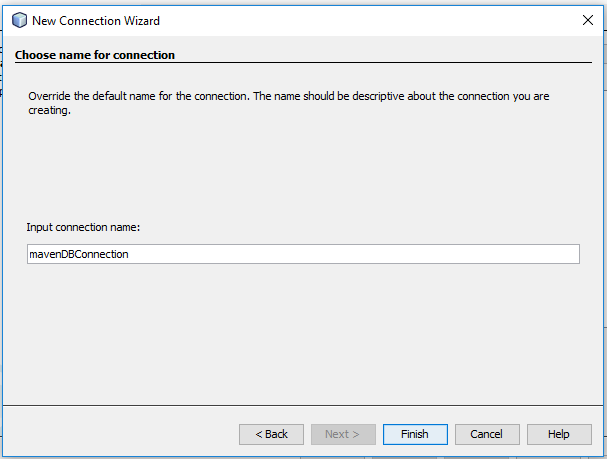


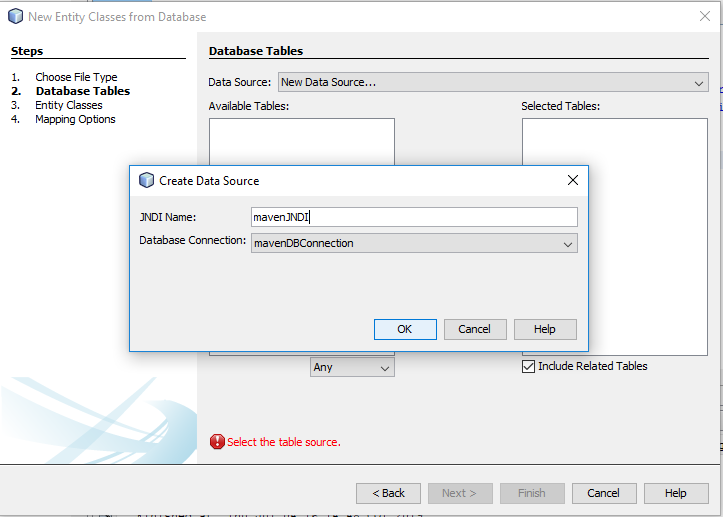


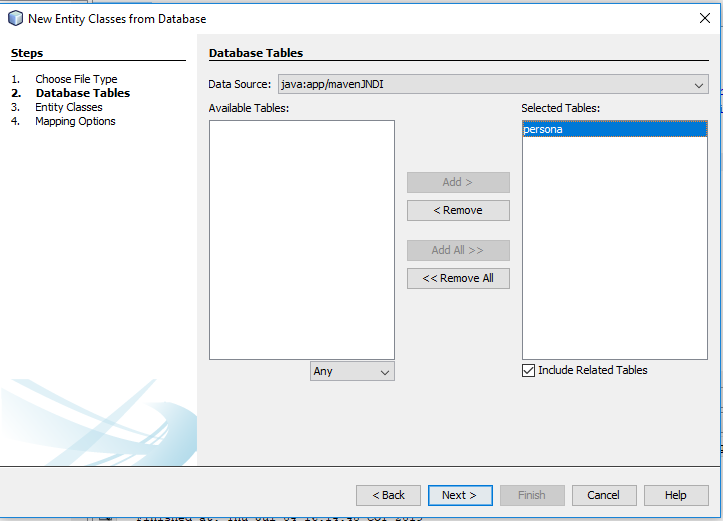


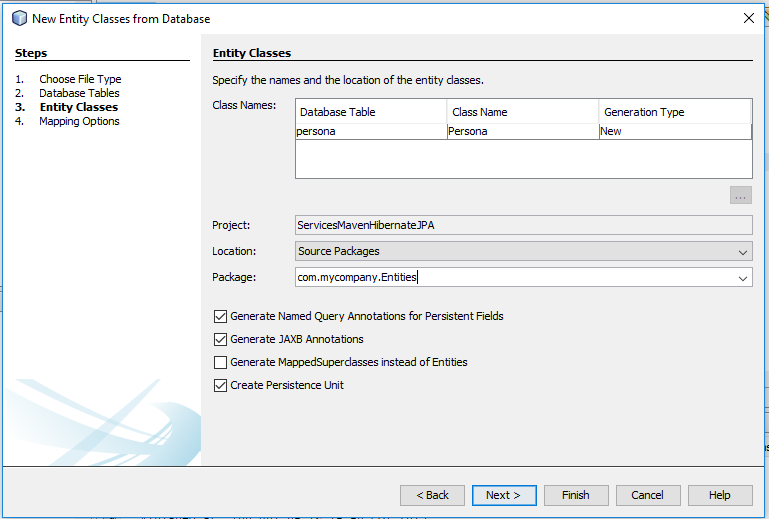


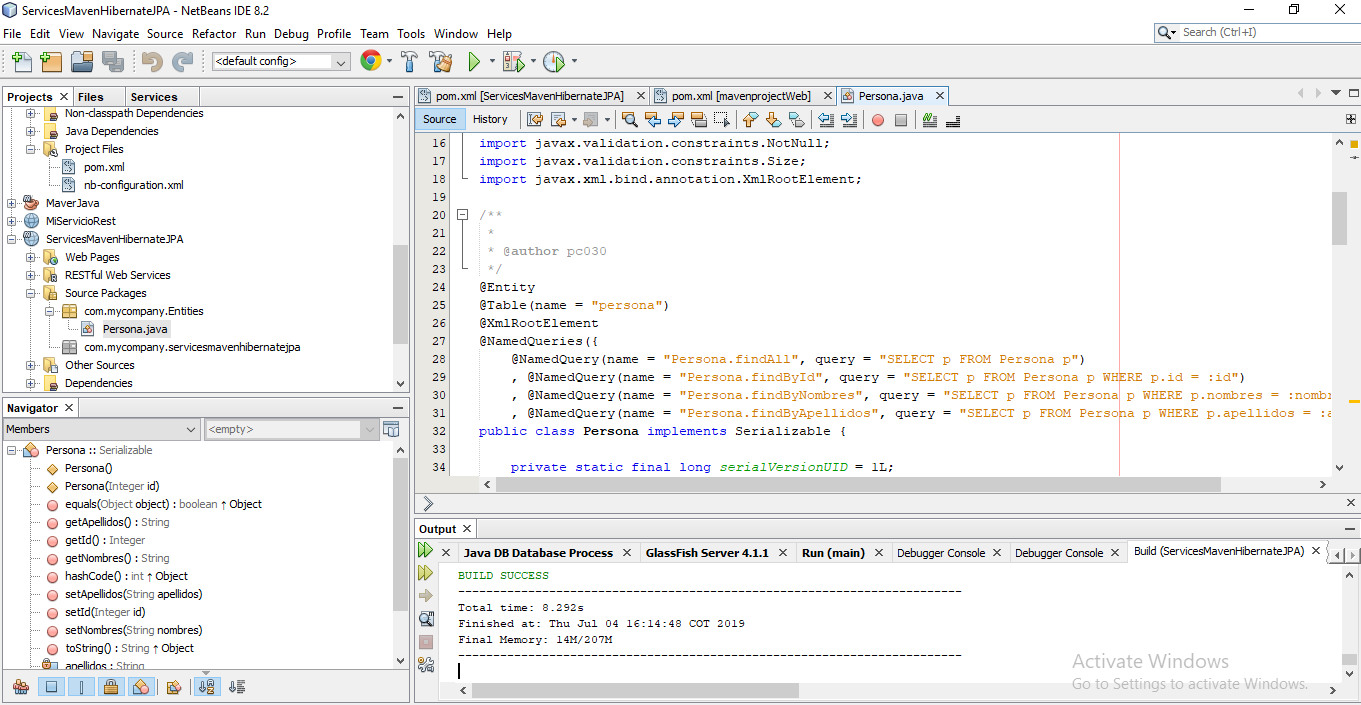




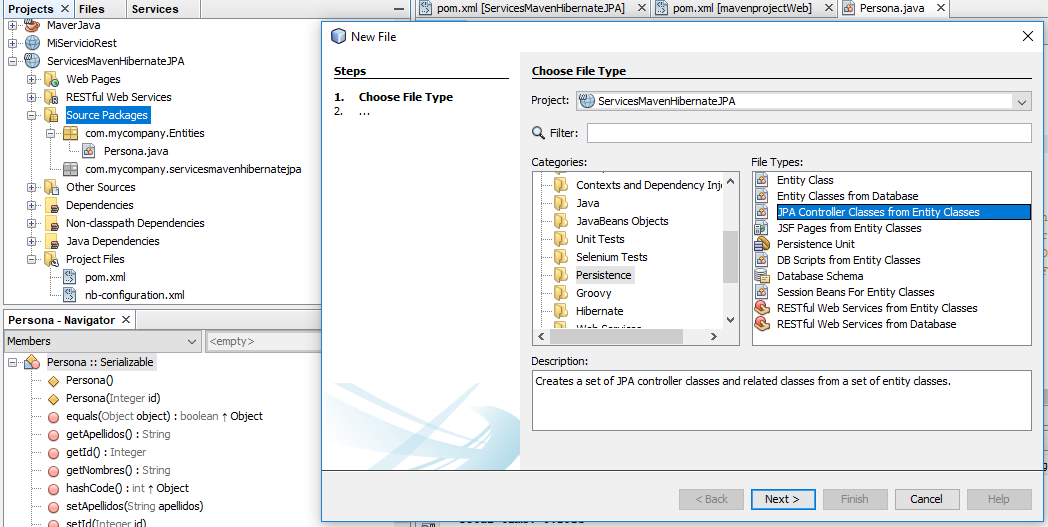


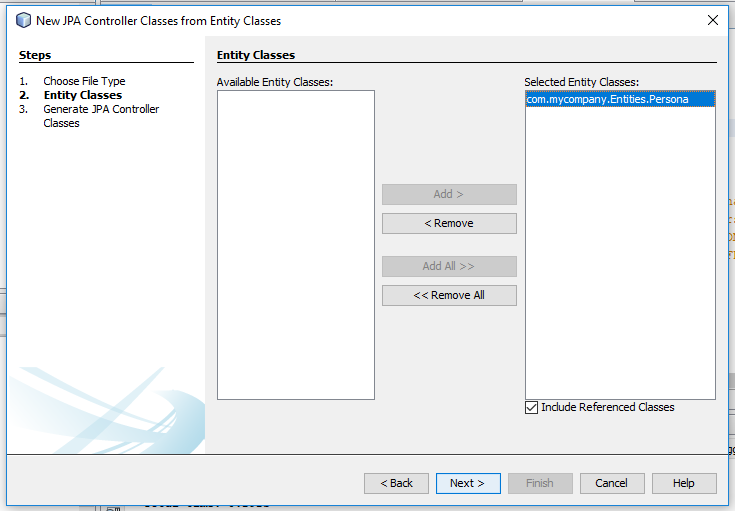


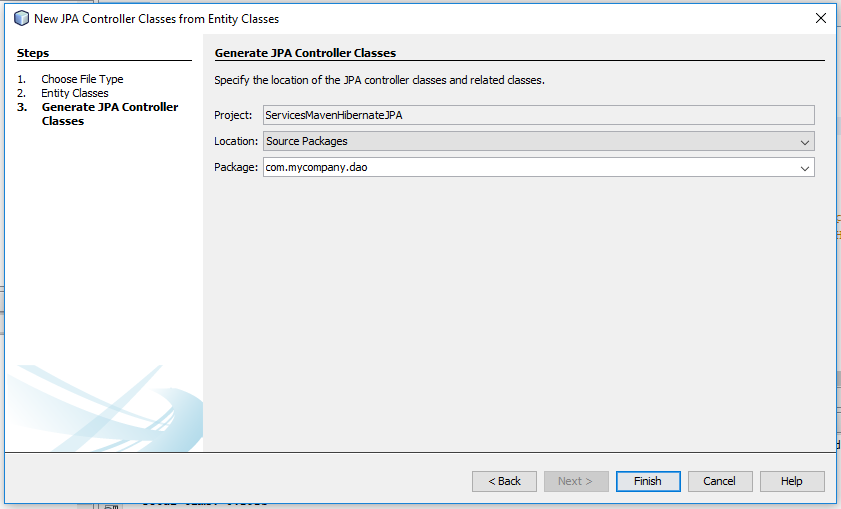


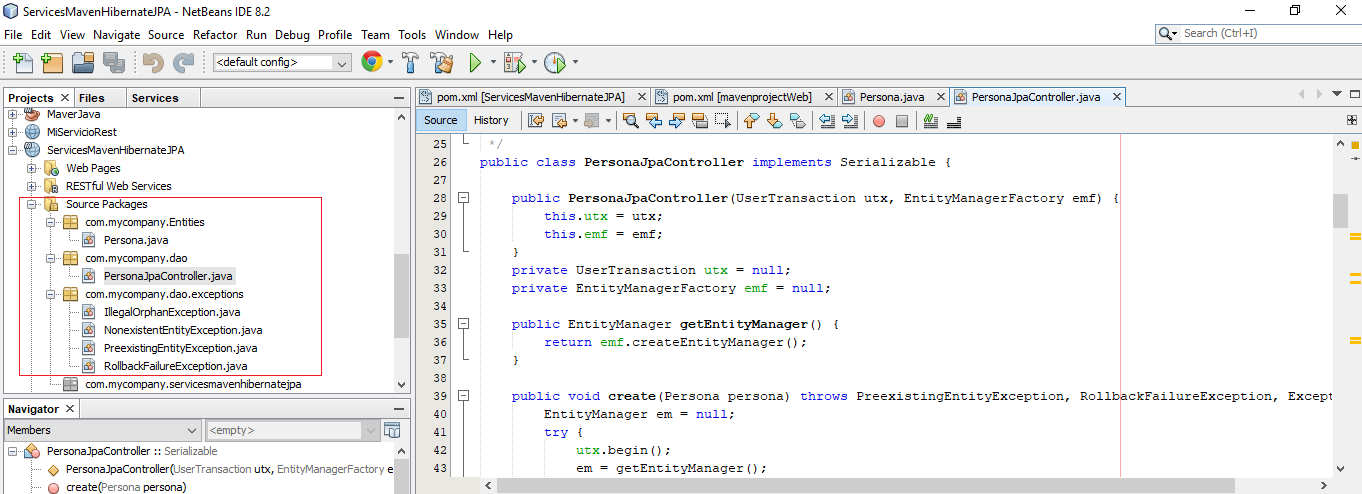


Vamos a crear El DAO para esta clase

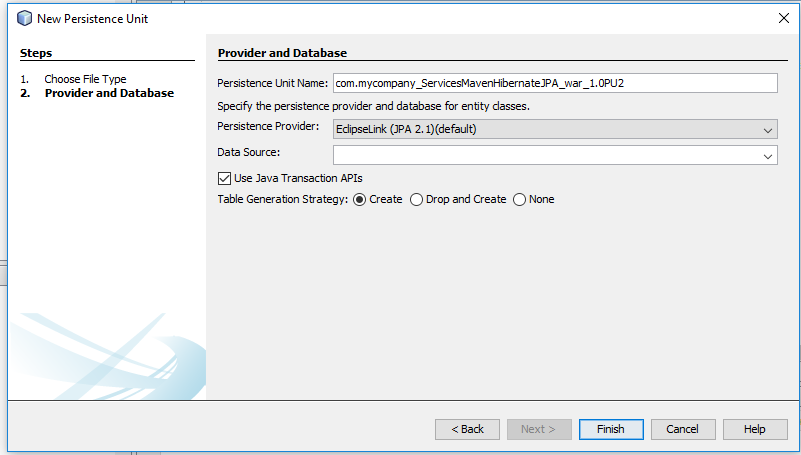


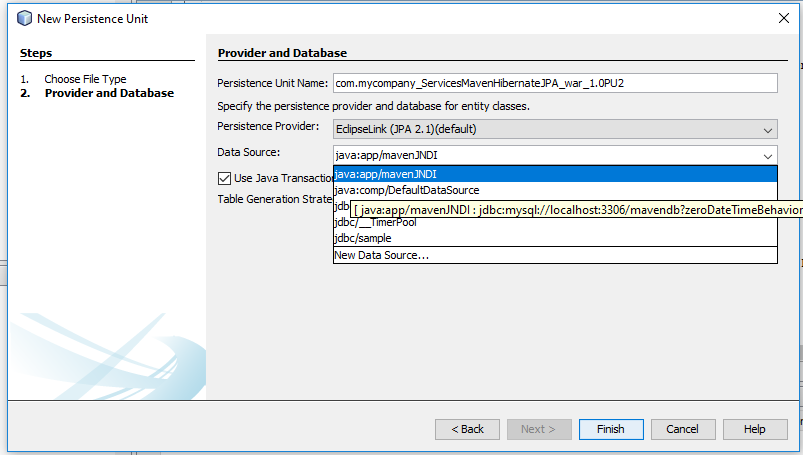


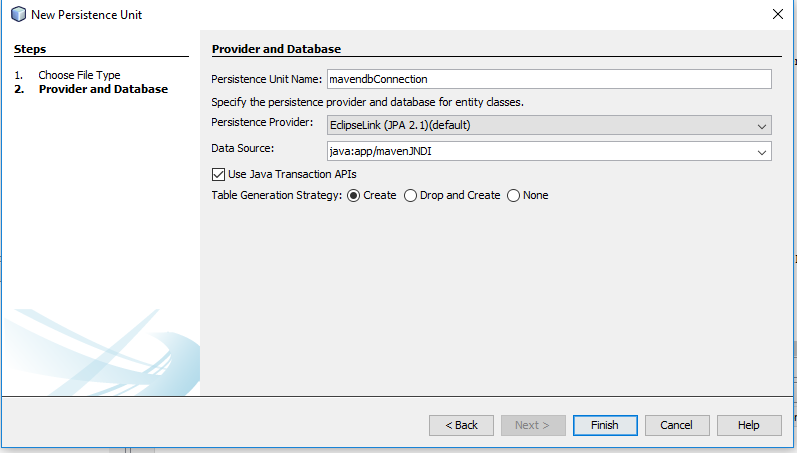




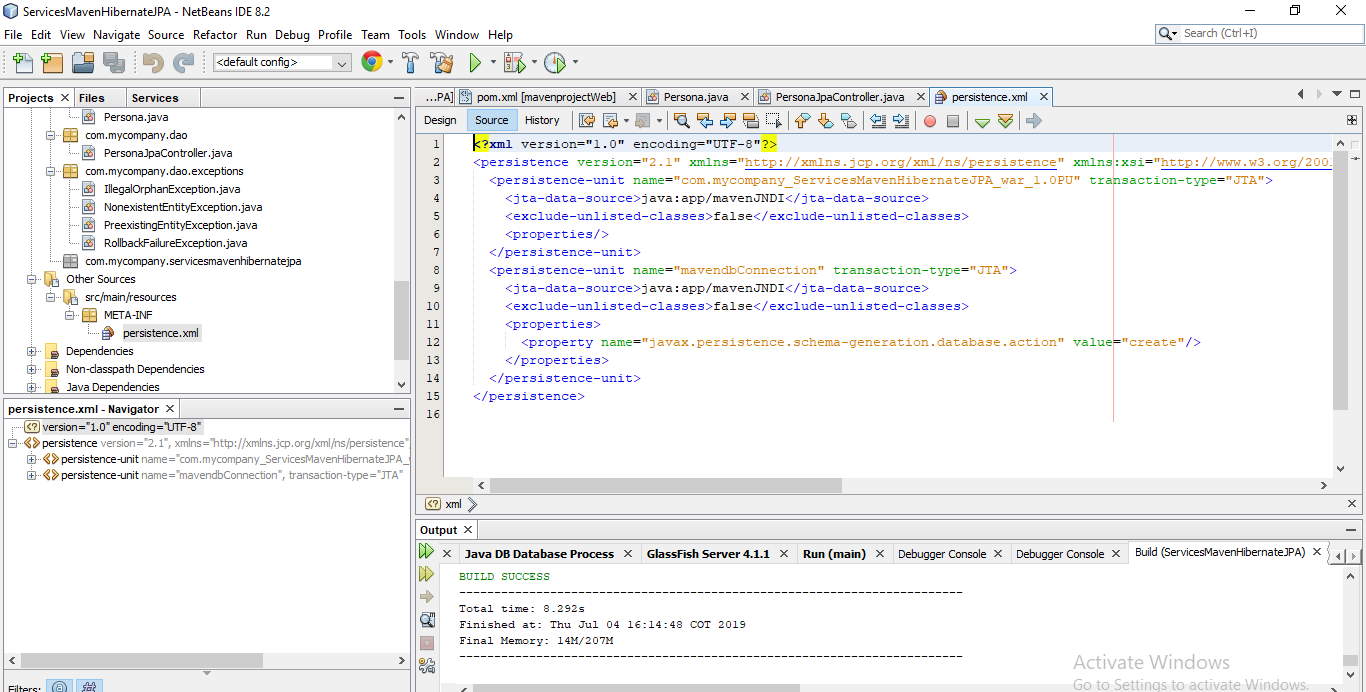
Ahora vamos a crear la persistencia que connecte nuestro JPA con Hibernate







Creara un archivo de persistencia en Other Source Meta-INF



Así como esta no me sirve para ello voy a tomar una plantilla para definir los datos de la bd

<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

<persistence-unit name="com.mycompany\_ServicesMavenHibernateJPA\_war\_1.0PU" transaction-type="JTA">

<jta-data-source>java:app/mavenJNDI</jta-data-source>

<exclude-unlisted-classes>false</exclude-unlisted-classes>

<properties/>

</persistence-unit>

<persistence-unit name="mavendbConnection" transaction-type="JTA">

<provider>org.hibernate.ejb.HibernatePersistence</provider>

<class>com.mycompany.Entity.Persona</class>

<exclude-unlisted-classes>true</exclude-unlisted-classes>

<!-- MySQL -->

<properties>

<property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/mavendb"/>

<property name="hibernate.connection.username" value="root"/>

<property name="hibernate.connection.password" value="root"/>

<property name="hibernate.connection.driver\_class" value="com.mysql.jdbc.Driver"/>

<property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>

<property name="hibernate.cache.provider\_class" value="org.hibernate.cache.NoCacheProvider"/>

<property name="hibernate.hbm2ddl.auto" value="update"/>

<!-- to prevent DB connection timeout (default MySQL timeout: 8h) -->

<property name="hibernate.c3p0.acquire\_increment" value="3" />

<property name="hibernate.c3p0.idle\_test\_period" value="14400" />

<property name="hibernate.c3p0.timeout" value="25200" />

<property name="hibernate.c3p0.max\_size" value="15" />

<property name="hibernate.c3p0.min\_size" value="3" />

<property name="hibernate.c3p0.max\_statements" value="0" />

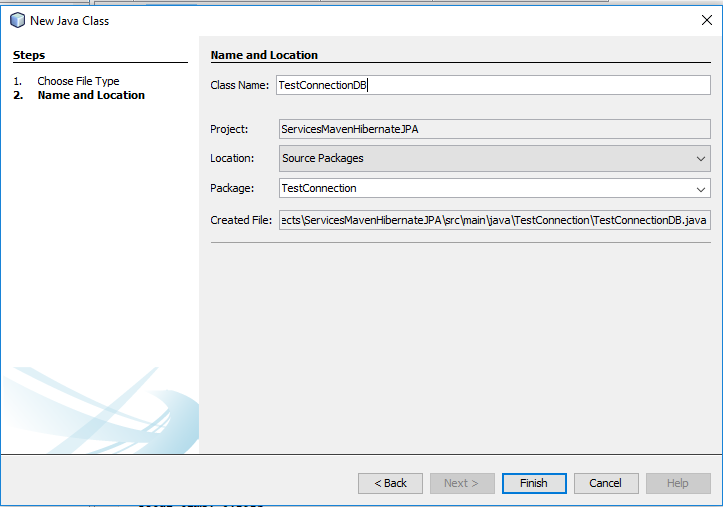
<property name="hibernate.c3p0.preferredTestQuery" value="select 1;" />

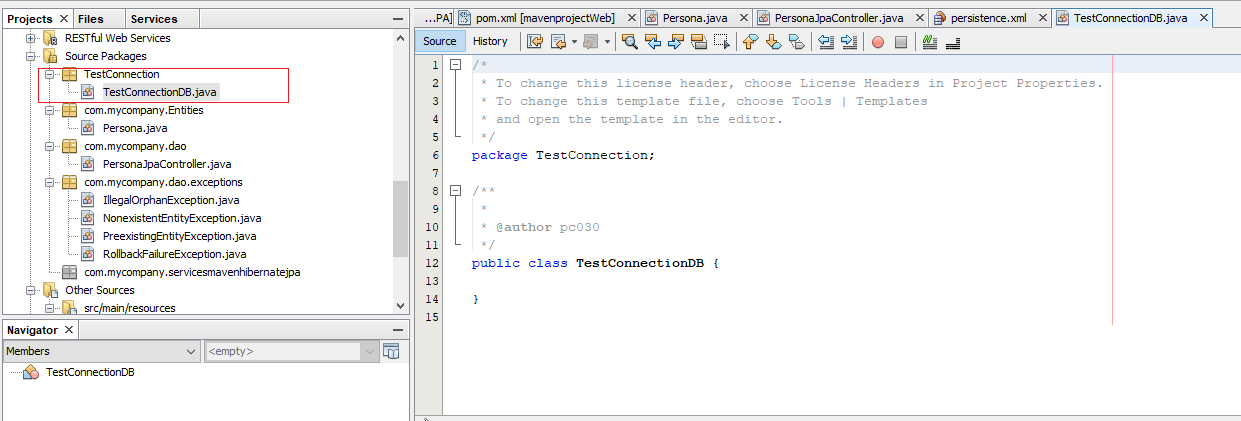
</properties>

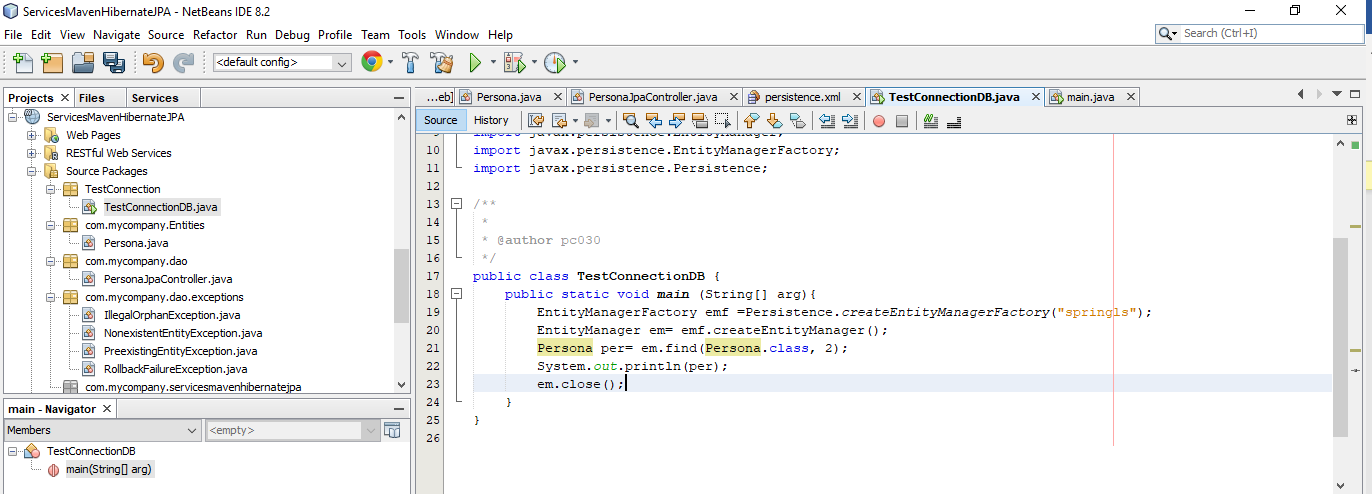
</persistence-unit>

</persistence>

Voy a crear una clase solo para probar si se estableció la conexión







/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package TestConnection;

import com.mycompany.Entities.Persona;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

/\*\*

\*

\* @author pc030

\*/

public class TestConnectionDB {

public static void main (String[] arg){

EntityManagerFactory emf =Persistence.createEntityManagerFactory("mavendbConnection");

EntityManager em= emf.createEntityManager();

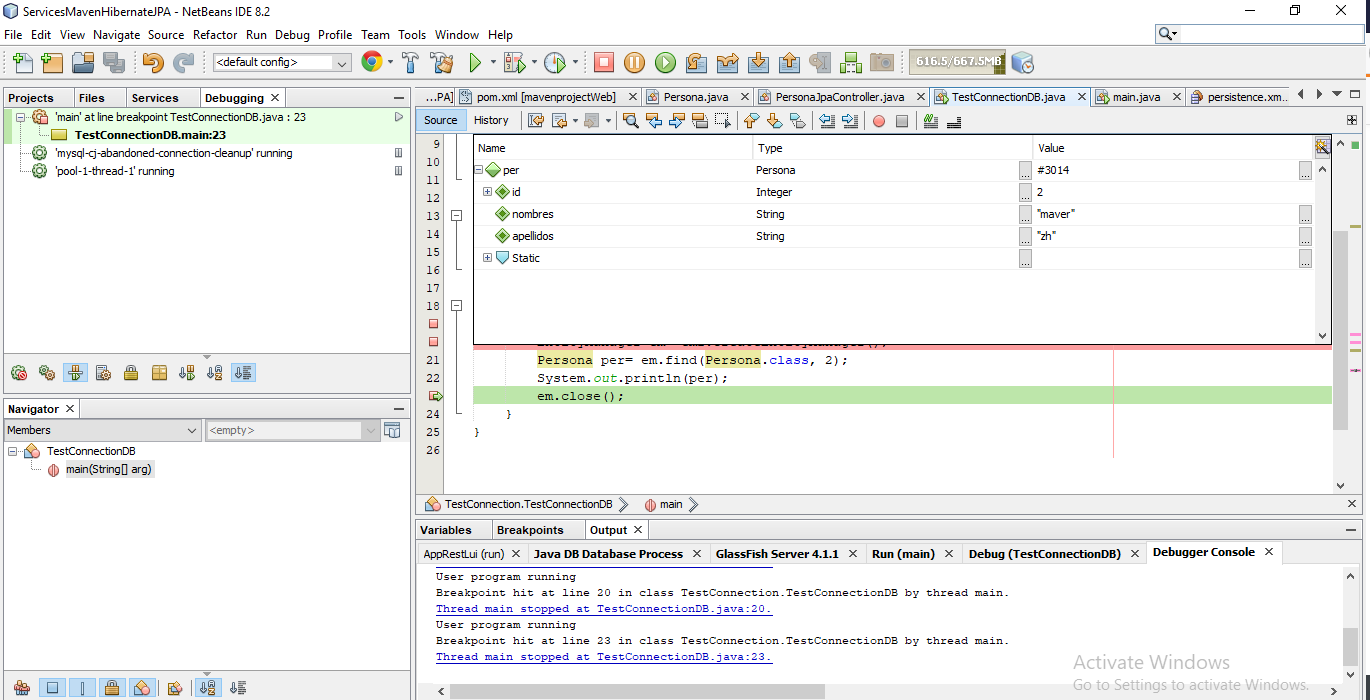
Persona per= em.find(Persona.class, 2);

System.out.println(per);

em.close();

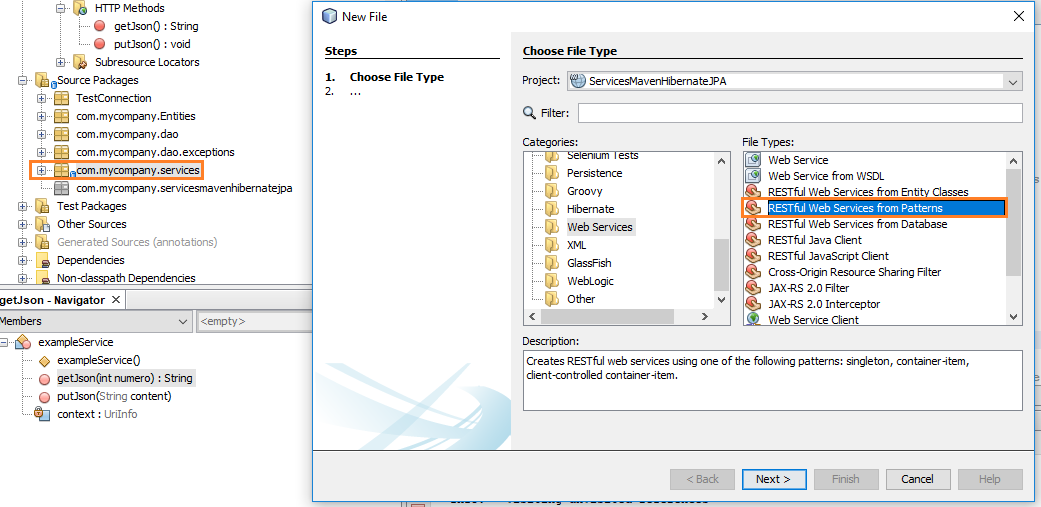
}

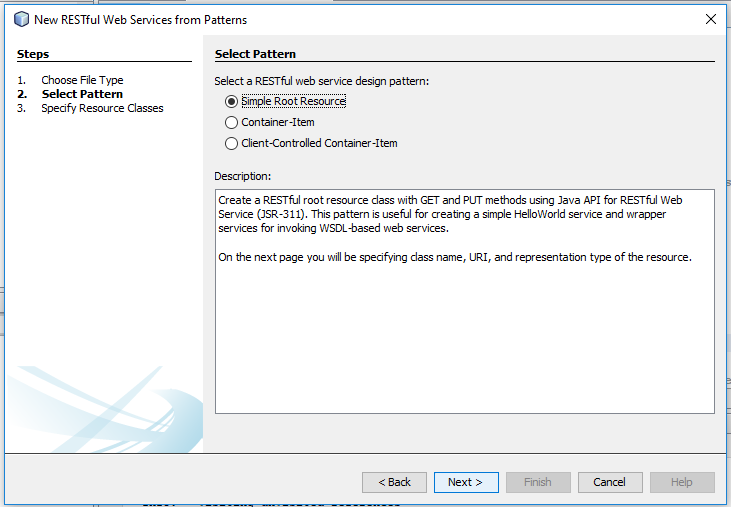
}

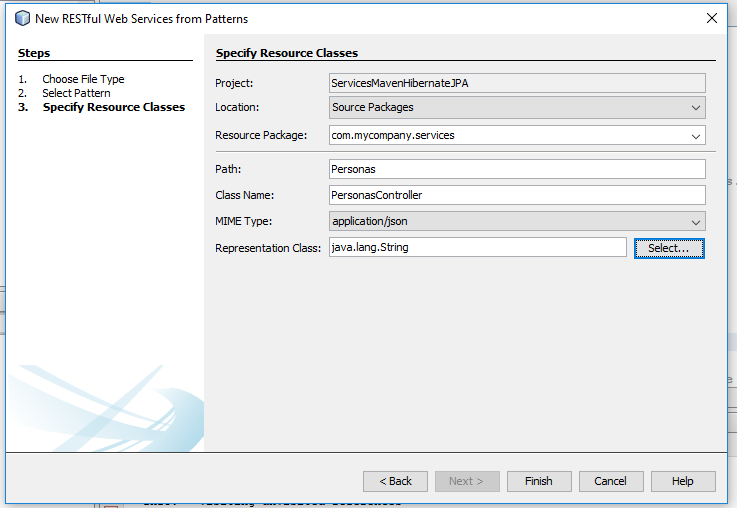


Ahora vamos a crear un servicio web

El paquete lo cree yo pero se puede poner cualquiera



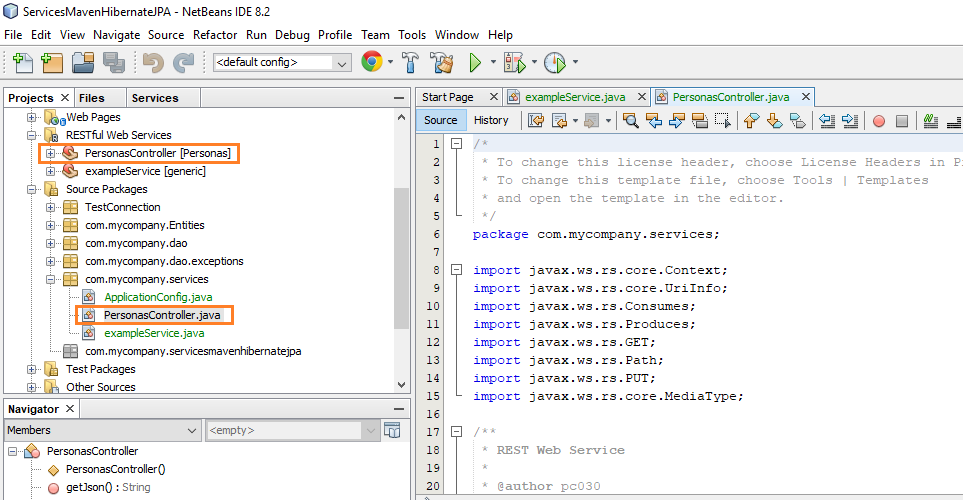




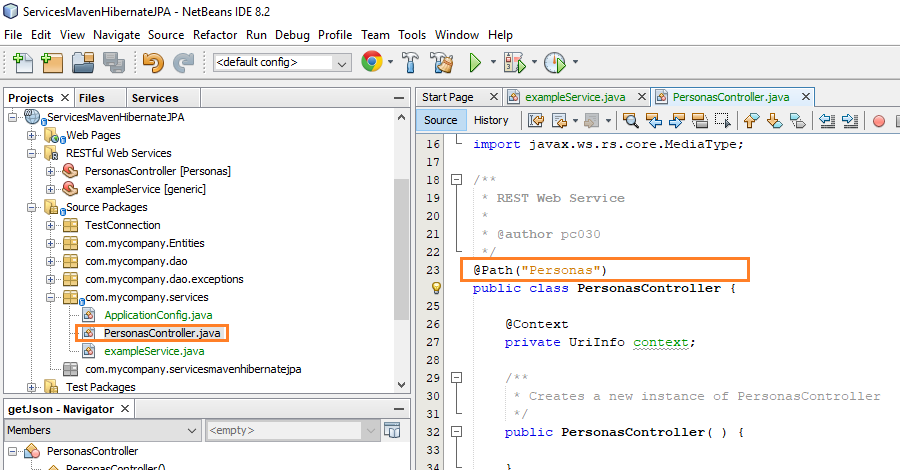
Path: Es el nombre de como se llamará nuestro servicio

ClassName: es el nombre de nuestra clase que se conectara en java

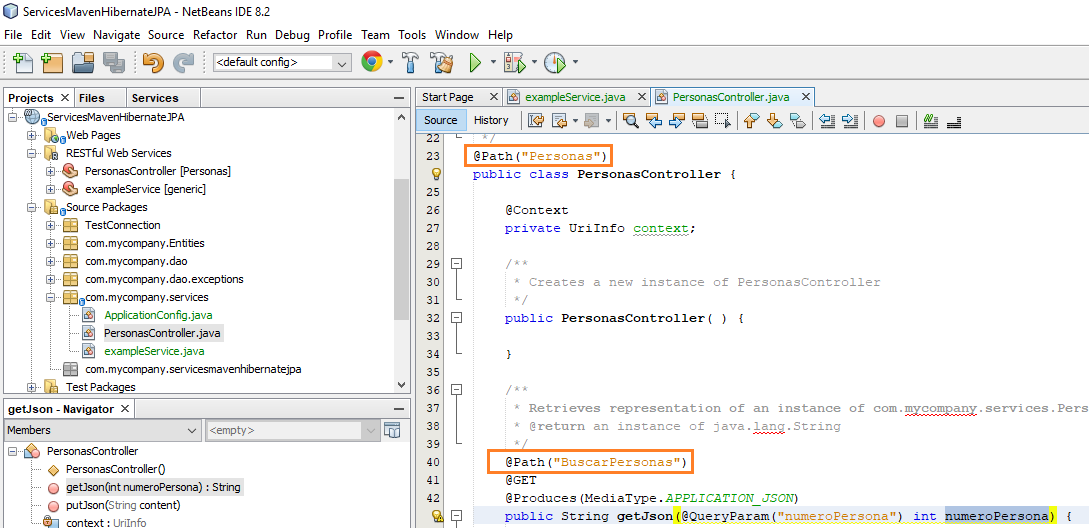
MineType: es el tipo de respuesta



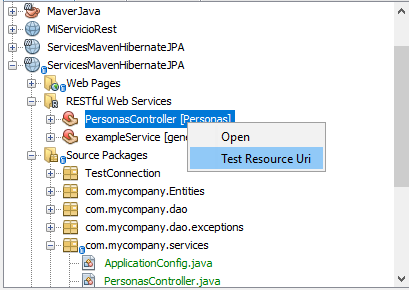
Dentro del controlador se creará el código path que es el nombre de nuestro servicio visto desde afuera



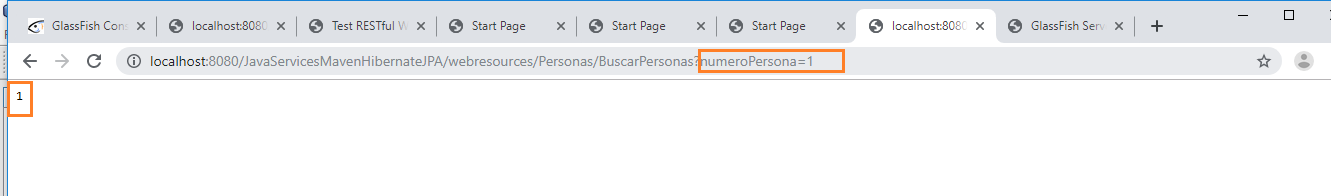
Ahora yo necesito también crear dentro de este servicio mis métodos para ello a cada método que necesite le voy a poner un path



Ahora para probar voy a hacer un test al servicio para ver en que url se publico



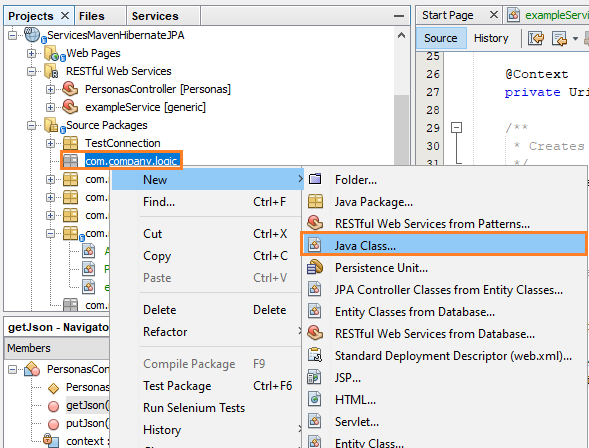
Tengo la raíz pero como yo defini un método con diferente nombre debo agregar el nombre

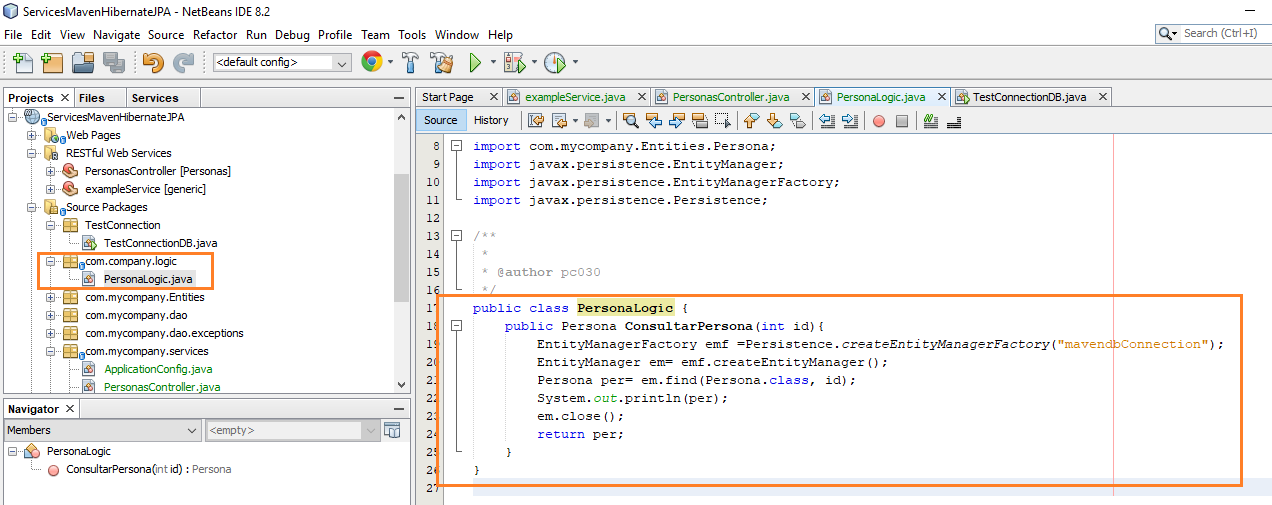


En la url tengo el nombre de mi servicio y el método dentro al que invoco

<http://localhost:8080/JavaServicesMavenHibernateJPA/webresources/Personas/BuscarPersonas?numeroPersona=1>

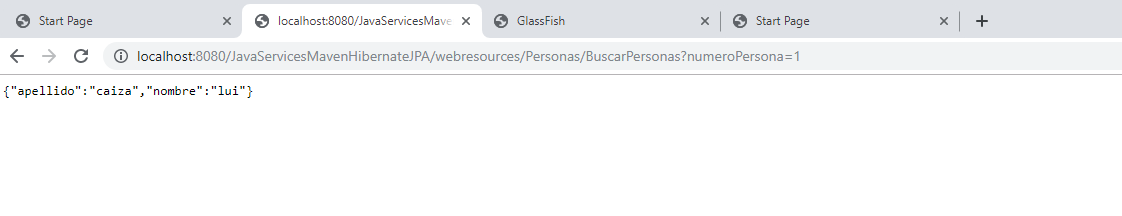
Ahora hagámoslo mas complejo para ello voy a crear una capa de lógica y dentro voy a hacer un objecto para devolver en el servicio.





Ahora lo voy a llamar desde mi servicio

Para que funcione usar glassfish mayor a 4.1.1 ya que esta versión cuenta con errores para que funcione use glassfish 5



Ahora vamos a jugar con los servicios Post.

Para ello normalmente enviamos y revivimos JSON vamos a crear un nuevo método para poder llamarlo como post

@Path("BuscarPersonasPost")

@POST

@Produces(MediaType.APPLICATION\_JSON)

@Consumes({MediaType.APPLICATION\_JSON})

public Response TraerPersonaPost(@Context HttpHeaders headers,pepito pep) {

//asi podemos tomar los parametros enviados en los headers

try{

String CodigoPlataforma = headers.getRequestHeader("CodigoPlataforma").get(0);

String Token = headers.getRequestHeader("Token").get(0);

//verifico token

if(!Token.equals("123"))

return Response.status(Response.Status.UNAUTHORIZED).entity("Acceso no Autorizado").build();

//Persona per= PersonaLogic.ConsultarPersona(persona.id());

pepito p = new pepito();

p.nombre="lui";

p.apellido="caiza";

return Response.ok(p, MediaType.APPLICATION\_JSON).build();

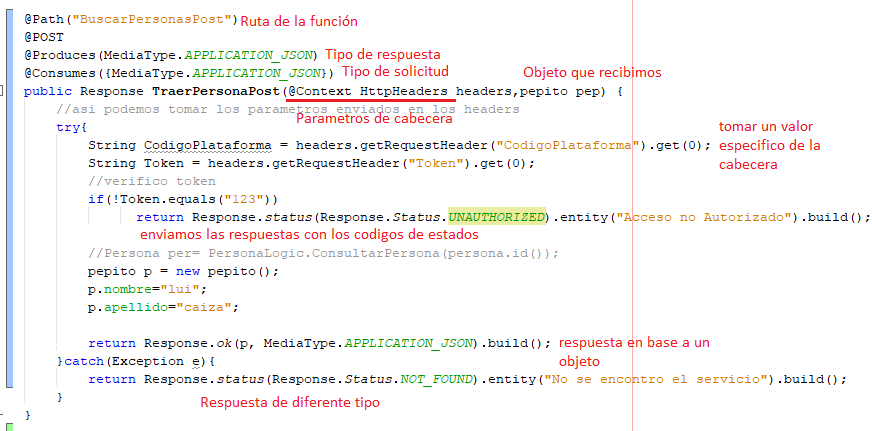
}catch(Exception e){

return Response.status(Response.Status.NOT\_FOUND).entity("No se encontro el servicio").build();

}

}

Descripción.



Lo podemos invocar con Swagger para lo enviaremos los parámetros

Post

http://localhost:8080/JavaServicesMavenHibernateJPA/webresources/Personas/BuscarPersonasPost

Host: localhost:8080

Connection: keep-alive

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9,es;q=0.8

Content-Length: 42

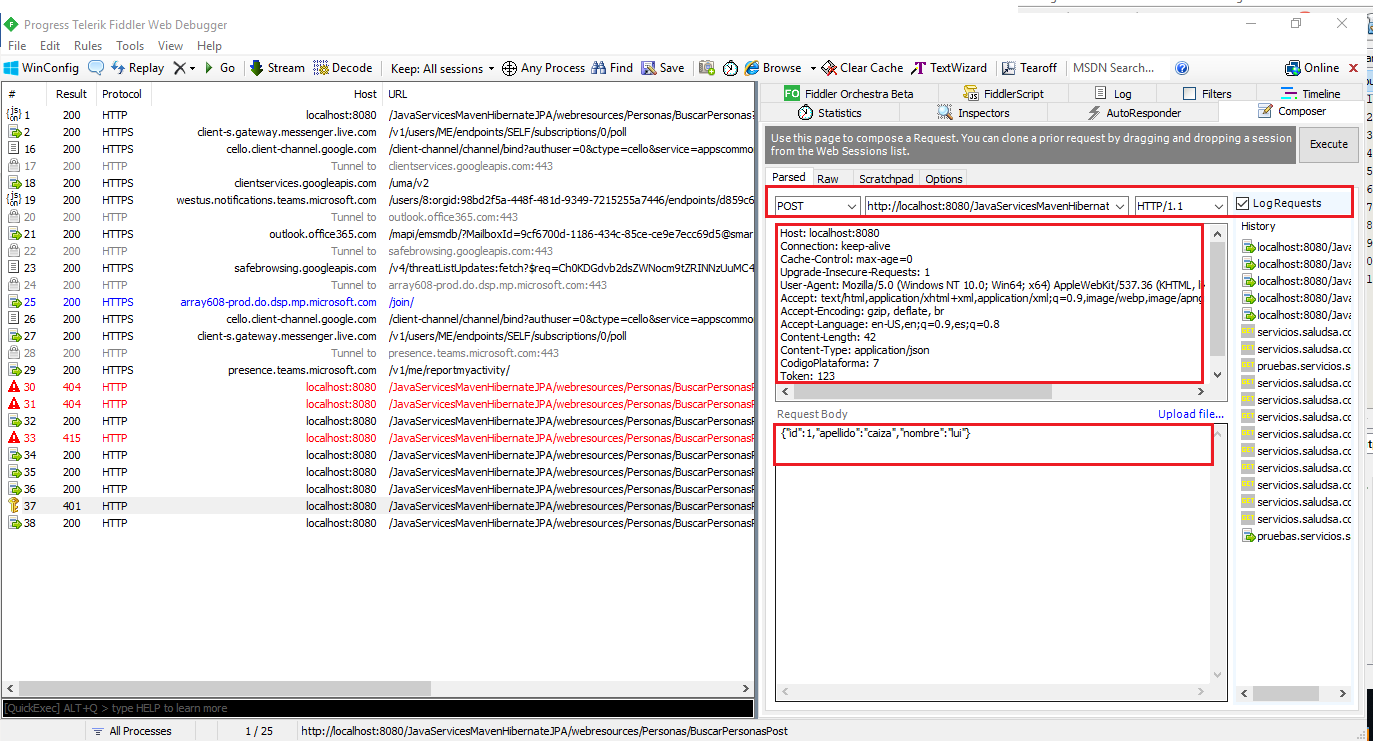
Content-Type: application/json

CodigoPlataforma: 7

Token: 123

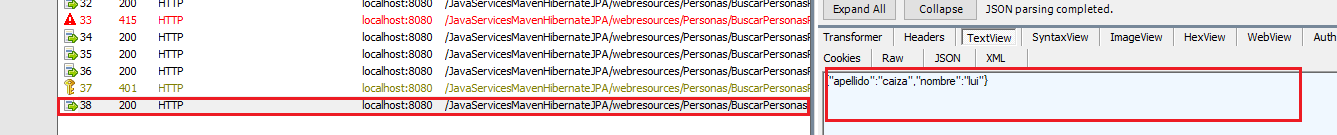
Objeto

{"id":1,"apellido":"caiza","nombre":"lui"}

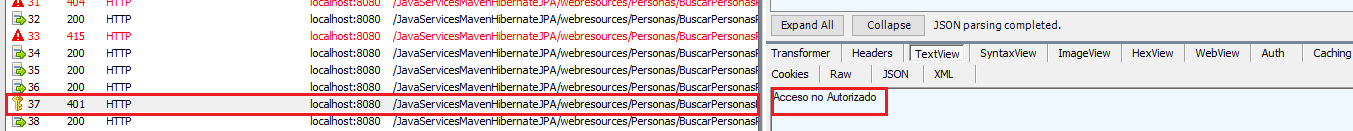


La respuesta dependerá de como la configuramos

Esta es la de ok



Esta por ejemplo es la que le puse de validación de token



Otro Ejemplo

<http://localhost:8080/JavaServicesMavenHibernateJPA/webresources/Login/ValidateUser>

Host: localhost:8080

Connection: keep-alive

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9,es;q=0.8

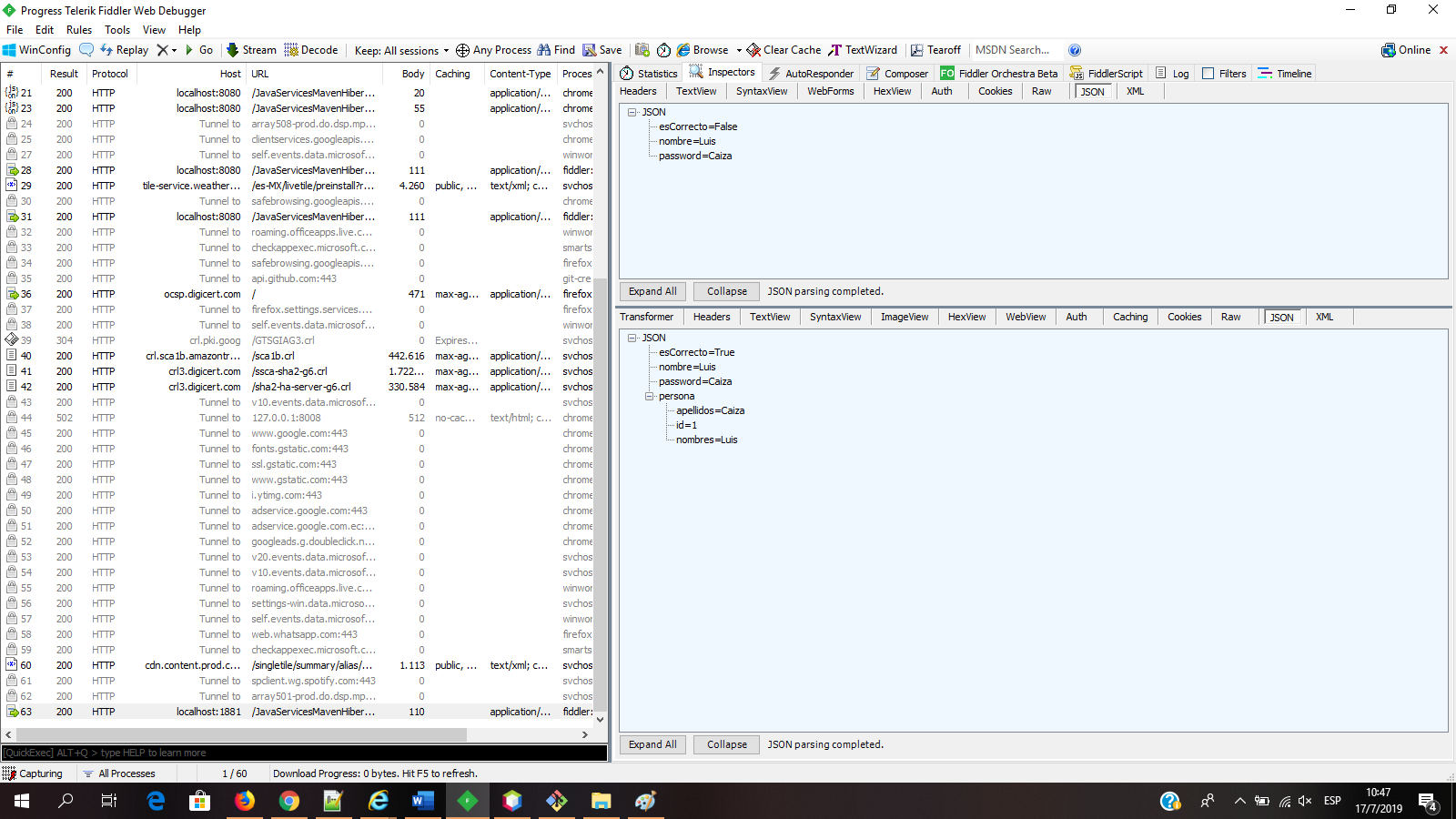
Content-Length: 55

Content-Type: application/json

CodigoPlataforma: 7

Token: 123

{"esCorrecto":false,"nombre":"Luis","password":"Caiza"}



Los estados cambian según mi programación, y estos serán leídos así en el cliente

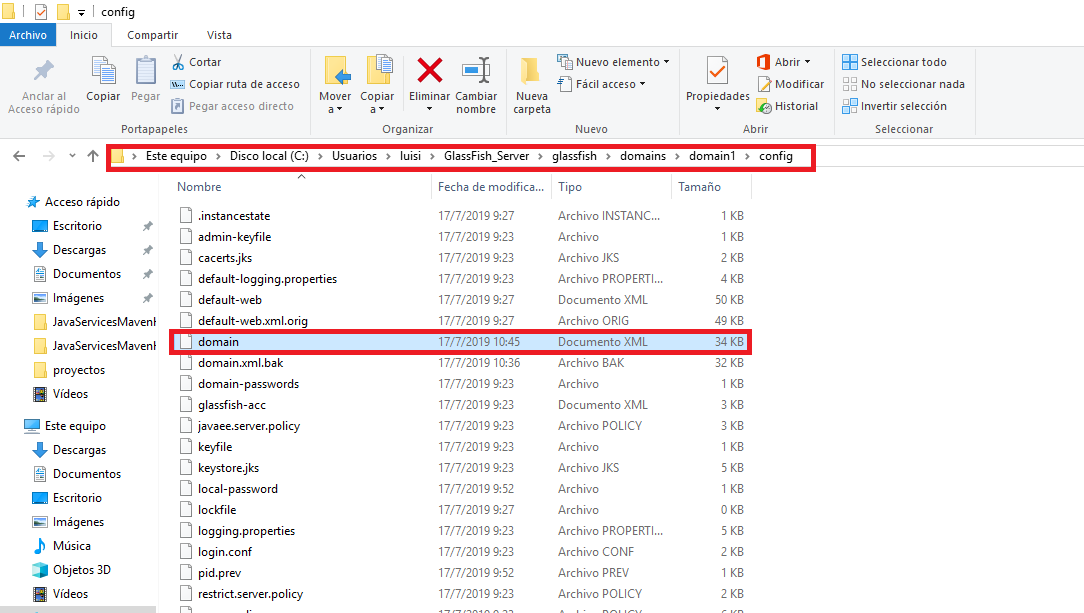
Si se desea cambiar el puerto en que se levanta glassfish hay que reconfigurarlo

D:\Programacion\Servidores\glassfish5\glassfish\domains\domain1\config



Ooo

C:\Users\luisi\GlassFish\_Server\glassfish\domains\domain1\config



Para permitir las connecciones cruzadas(Permitir que desde otro servidor se conecten a nuestro servicio vamos a configurar nuestro proyecto)

1. Agregar las siguientes dependencias

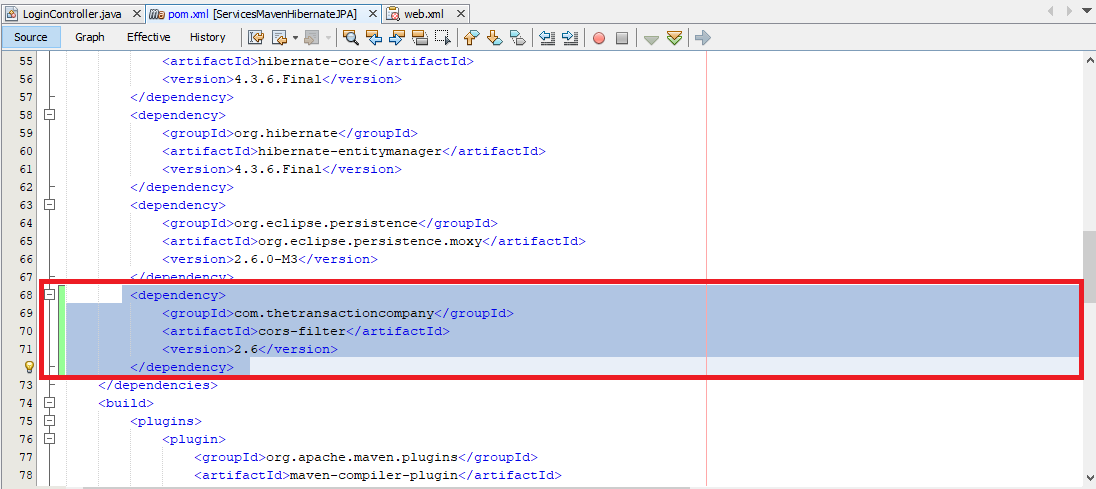
<dependency>

<groupId>com.thetransactioncompany</groupId>

<artifactId>cors-filter</artifactId>

<version>2.6</version>

</dependency>



Agregamos el web.xml

<filter>

<filter-name>CORS</filter-name>

<filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>

<init-param>

<param-name>cors.supportedMethods</param-name>

<param-value>GET, POST, HEAD, PUT, PATCH, DELETE, OPTIONS</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CORS</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

