A dark blue vertical bar runs along the left edge of the page. A blue arrow-shaped banner points to the right from this bar, containing the university's name. In the bottom-left corner, several thin, curved lines in dark blue and light gray sweep upwards and to the right.

Universidad Peruana de  
Ciencias Aplicadas

# Implementando Web Services

Guía Tutorial - Java

Roy Alejo Taza Rojas  
SOFTWARE FACTORY

## Contenido

INTRODUCCION .....	2
DESARROLLO .....	2
Proyecto Acceso a Base de datos.....	2
Web Services – Rest .....	12
Listar .....	17
Obtener .....	18
Insertar .....	19
Actualizar.....	20
Eliminar .....	20
Web Services – SOAP .....	22
CRUD completo .....	23
Probando.....	25

## INTRODUCCION

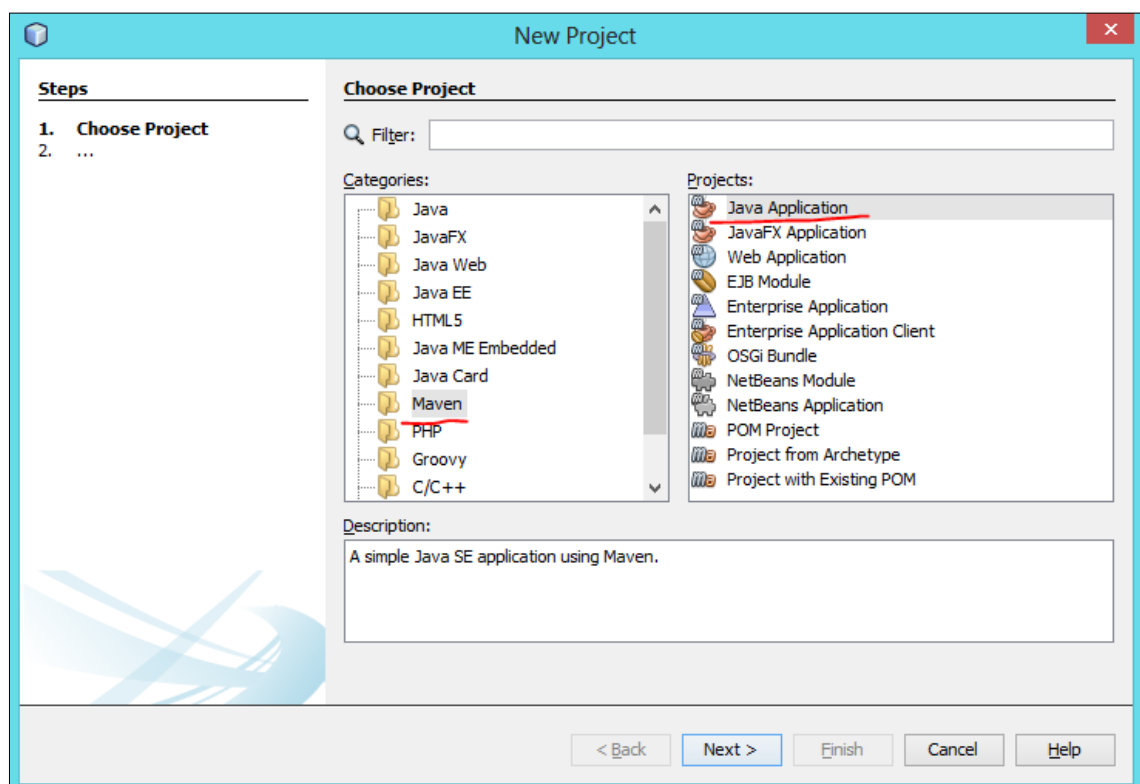
Seguro que habrás oído el bombardeo publicitario, y probablemente estarás aturrido con todos los acrónimos. Por eso ¿qué son los Servicios Web y cómo podemos utilizarlos? Este tutorial intentará desmitificar los Servicios Web y mostrará, paso a paso, cómo construirlos, desplegarlos, usarlos y encontrarlos.

Los servicios web son una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares. Es una máquina que atiende las peticiones de los clientes web y les envía los recursos solicitados.

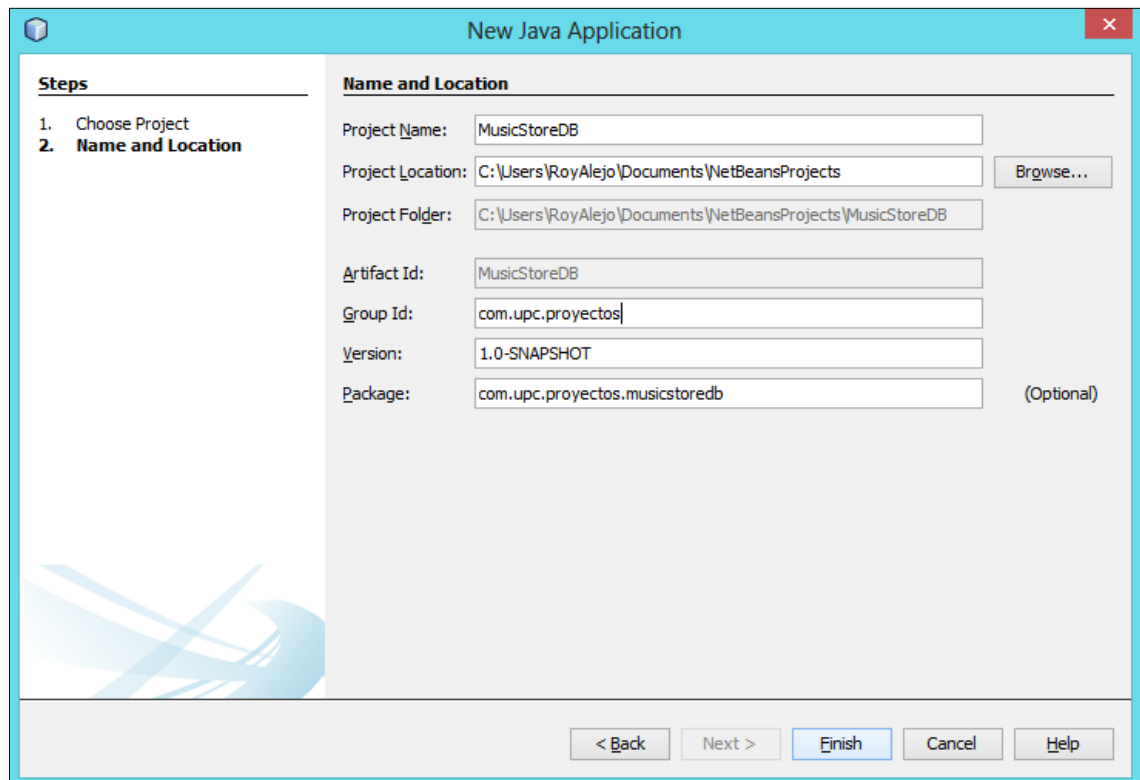
## DESARROLLO

### Proyecto Acceso a Base de datos

Crearemos un proyecto Maven -> Java Application. Este proyecto nos servirá para manejar toda la lógica de conexión a la base de datos MySQL.



Nombraremos al proyecto **MusicStoreDB** y al Group Id **com.upc.proyectos** y daremos clic en Finish.



Ahora usaremos Hibernate con anotaciones para conectarnos a la Base de Datos. Si viste nuestra guía anterior te será fácil entender. Sino, puedes revisarla en este [ENLACE](#).

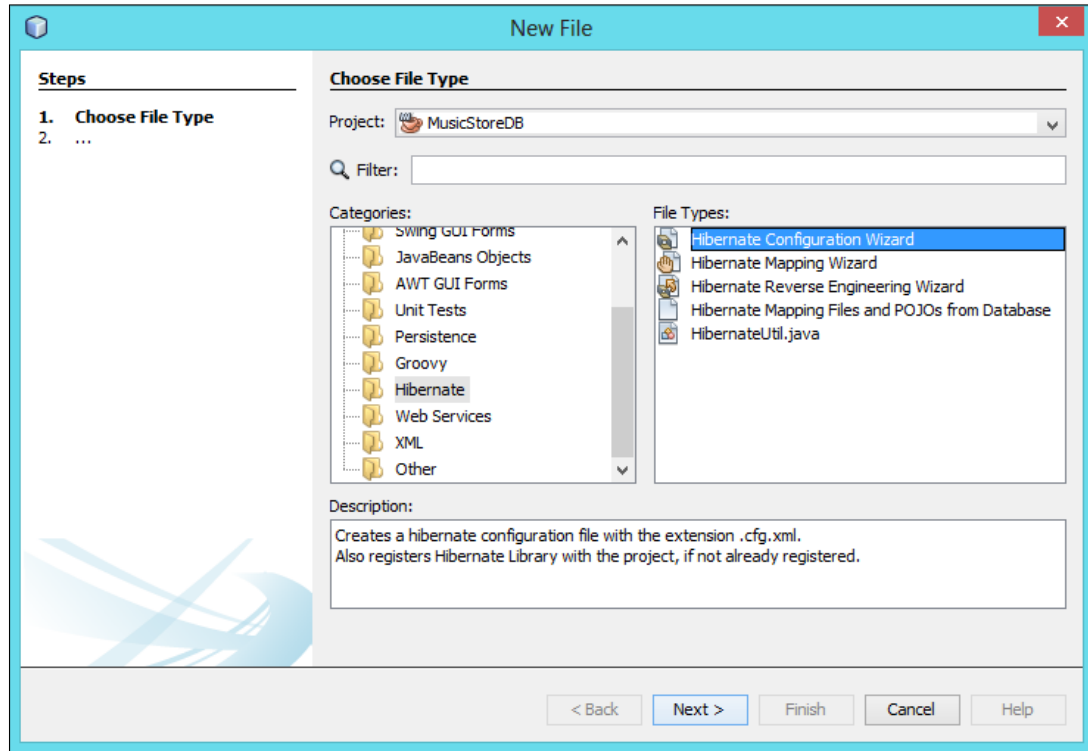
Ya que esta guía trata de Web Services, no entraremos a tanto de detalle en los parámetros de configuración. Sin embargo, para los que revisaron la guía anterior, este parte les servirá para refrescar conocimientos y conocer una manera más rápida de configurar y usar Hibernate.

Las dependencias que usará nuestro proyecto serán hibernate-entitymanager, hibernate-core y mysql-connector-java. Las dependencias de nuestro archivo **pom.xml** quedarían así:

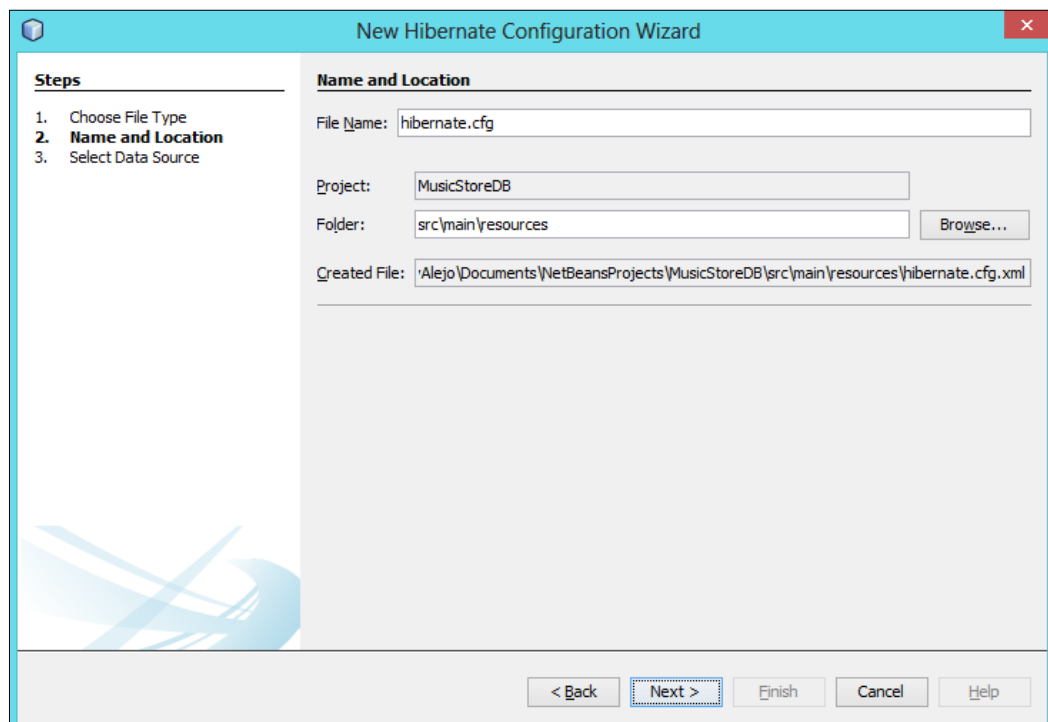
```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.1.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.1.Final</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.23</version>
  </dependency>
</dependencies>
```

</dependencies>

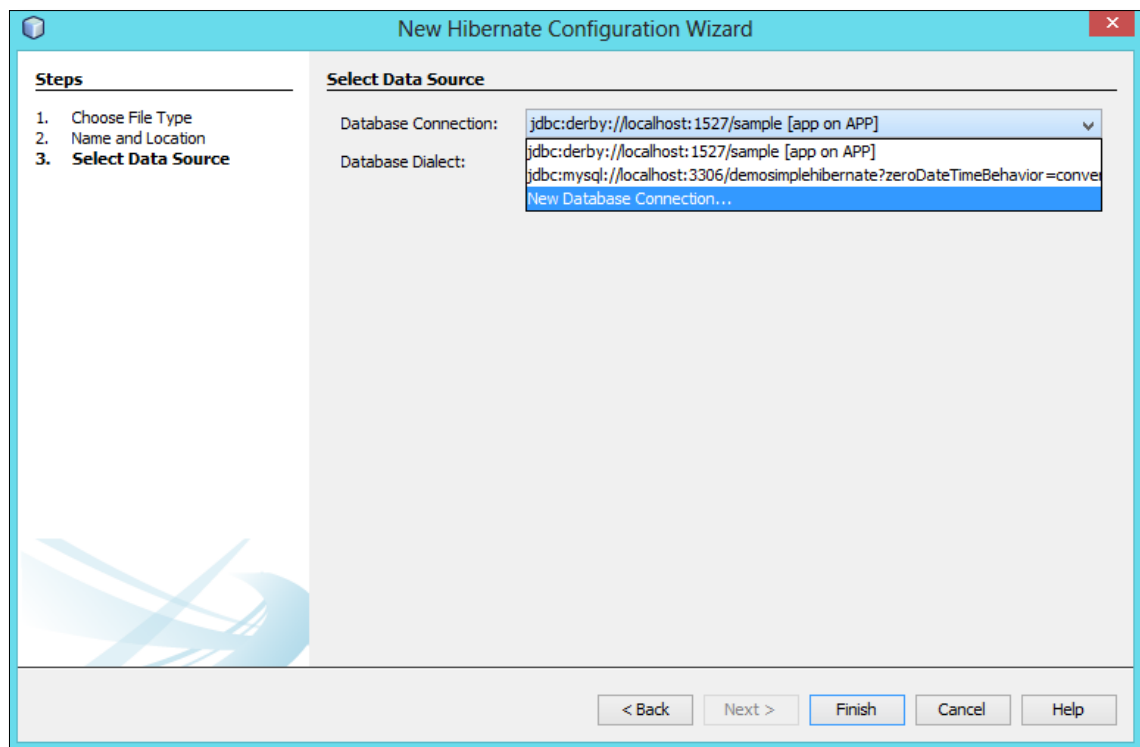
Hacemos clic derecho en nuestro proyecto: New -> Other... -> seleccionamos **Hibernate Configuration Wizard** y clic en Next.



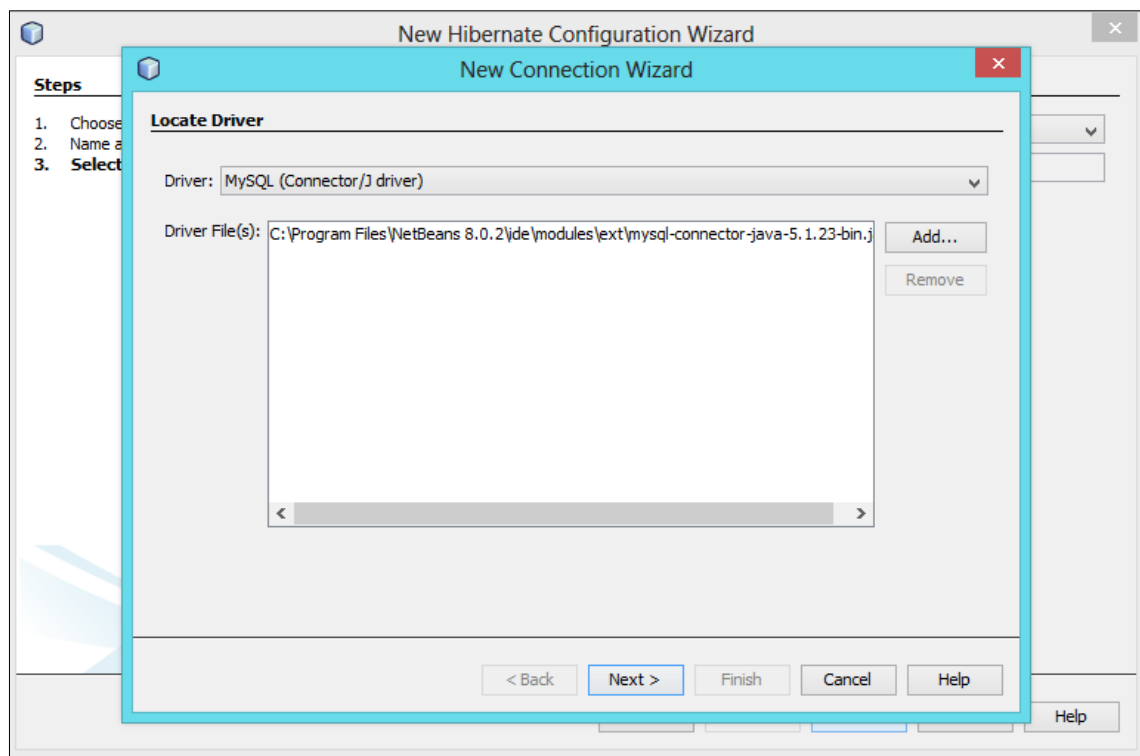
Dejamos el nombre del archivo y folder por defecto. Next.



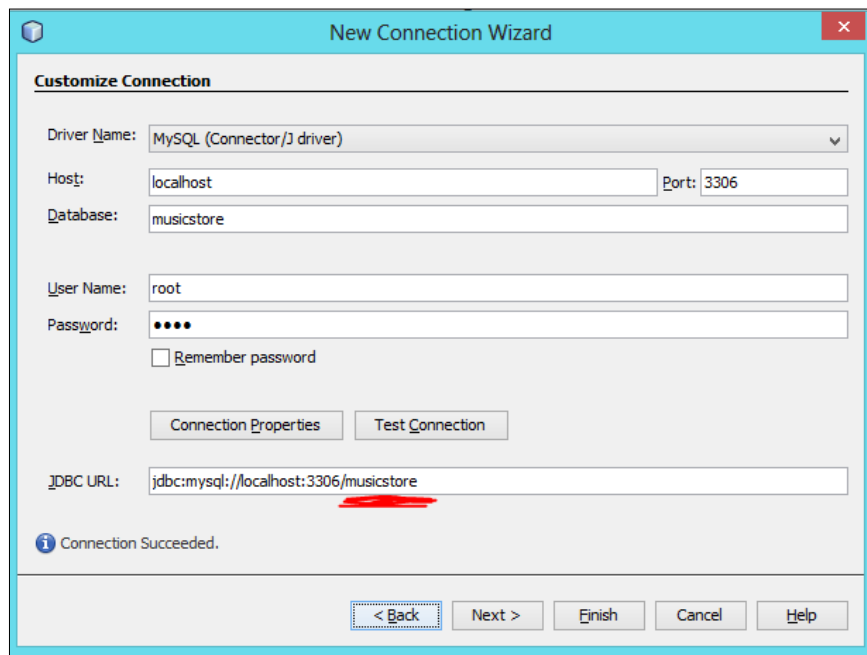
Seleccionamos **New Database Connection...** como muestra la imagen.



Se abrirá una ventana New Connection Wizard. Seleccionamos como **Driver a MySQL (Connector/J driver)**. Next.



Configuramos nuestra conexión a nuestra base de datos. Asimismo, acá indicamos el nombre de la base de datos, que en nuestro caso será **musicstore**. Podemos probar la conexión haciendo clic en Test Connection. Click en Finish.

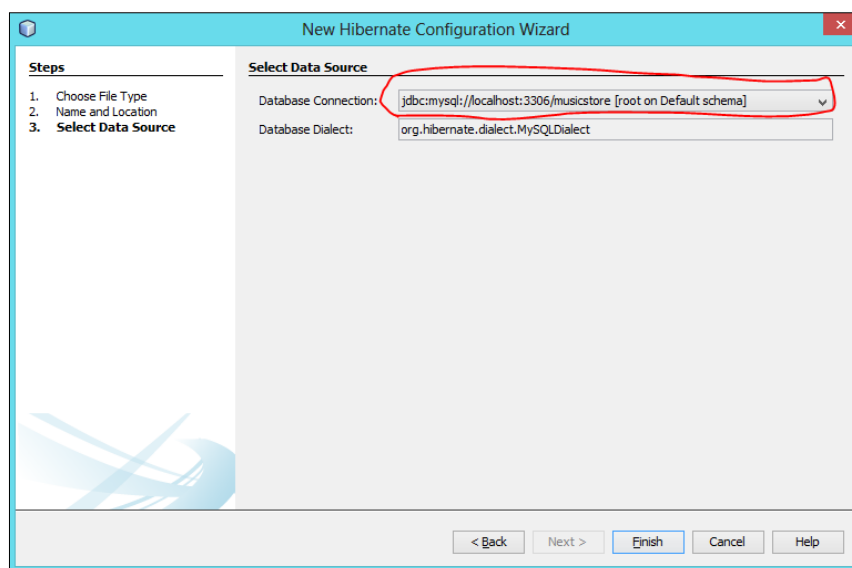


Nota: crear la base de datos musicstore en MySQL y la tabla disco, puedes ejecutar este script:

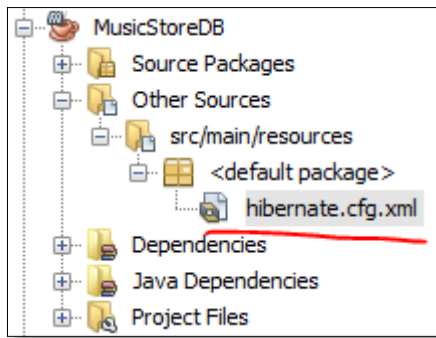
```
create database musicstore;
```

```
CREATE TABLE `disco` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `nombre` varchar(255) DEFAULT NULL,  
  `autor` varchar(255) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;
```

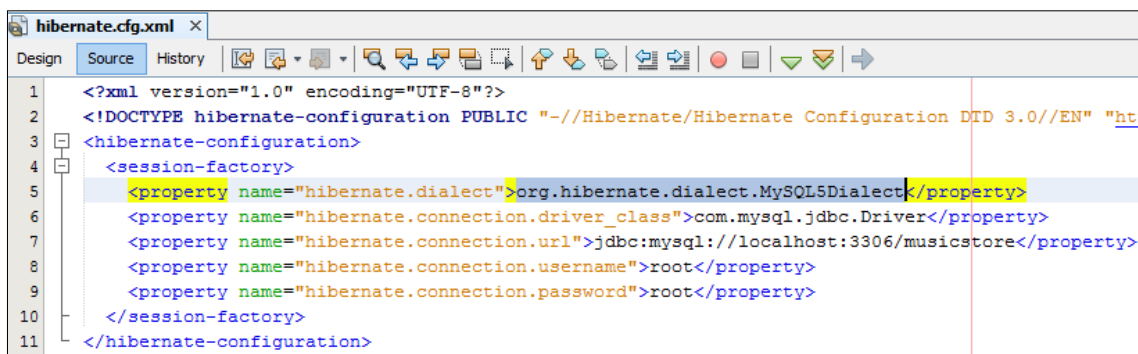
Clic en Finish:



Nuestro archivo de configuración se ha creado:



Por defecto, el dialecto que se asigna es org.hibernate.dialect.MySQLDialect, lo cambiaremos a org.hibernate.dialect.MySQL5Dialect



¡A este archivo de configuración sólo faltaría agregar las clases con anotaciones!

Crearemos un paquete **com.upc.proyectos.musicstoredb.entidades** y agregaremos una clase **Disco**, la cual tendrá los atributos id, nombre y autor. Agregando las anotaciones quedaría de la siguiente manera:

```
package com.upc.proyectos.musicstoredb.entidades;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Disco implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String nombre;
    private String autor;

    public Disco() {
    }
}
```



```

public Disco(String nombre, String autor) {
    this.nombre = nombre;
    this.autor = autor;
}

public long getId() {
    return id;
}

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

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getAutor() {
    return autor;
}

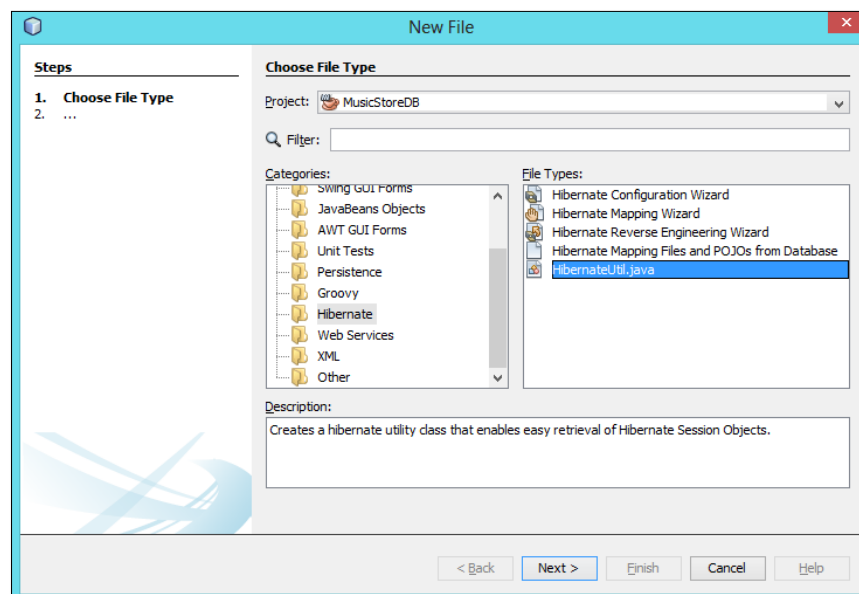
public void setAutor(String autor) {
    this.autor = autor;
}
}

```

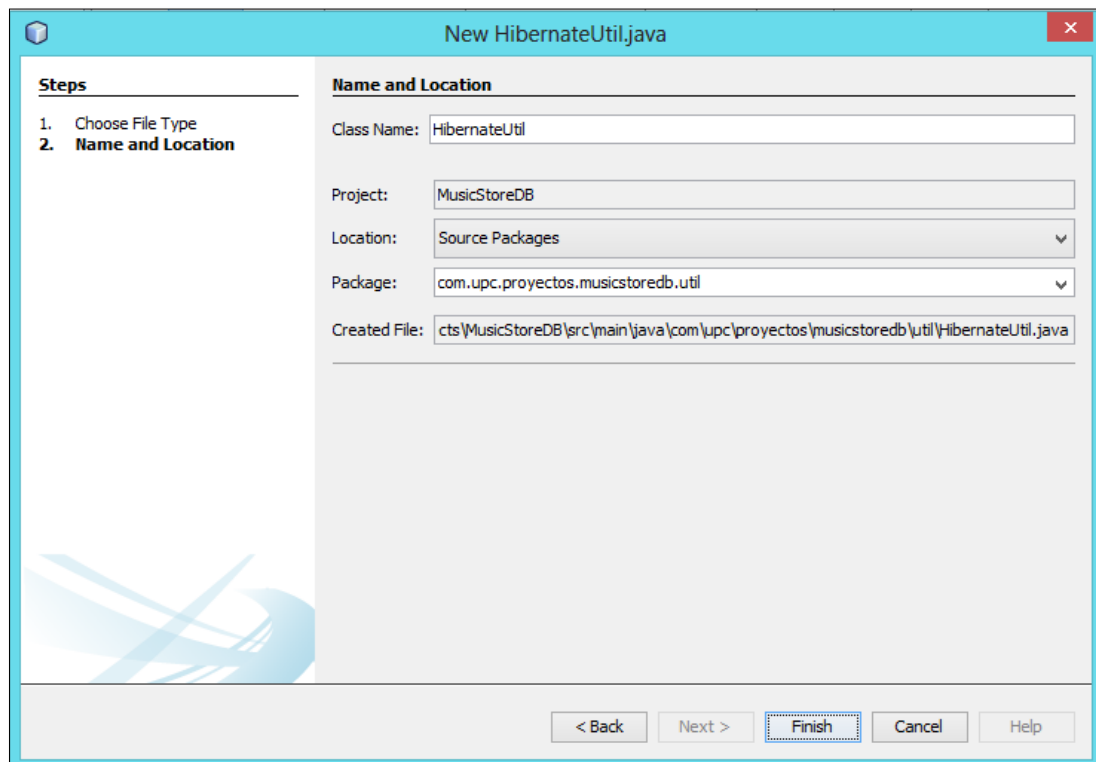
**Mapeamos** nuestra entidad en **hibernate.cfg** con lo siguiente:

```
<mapping class="com.upc.proyectos.musicstoredb.entidades.Disco" />
```

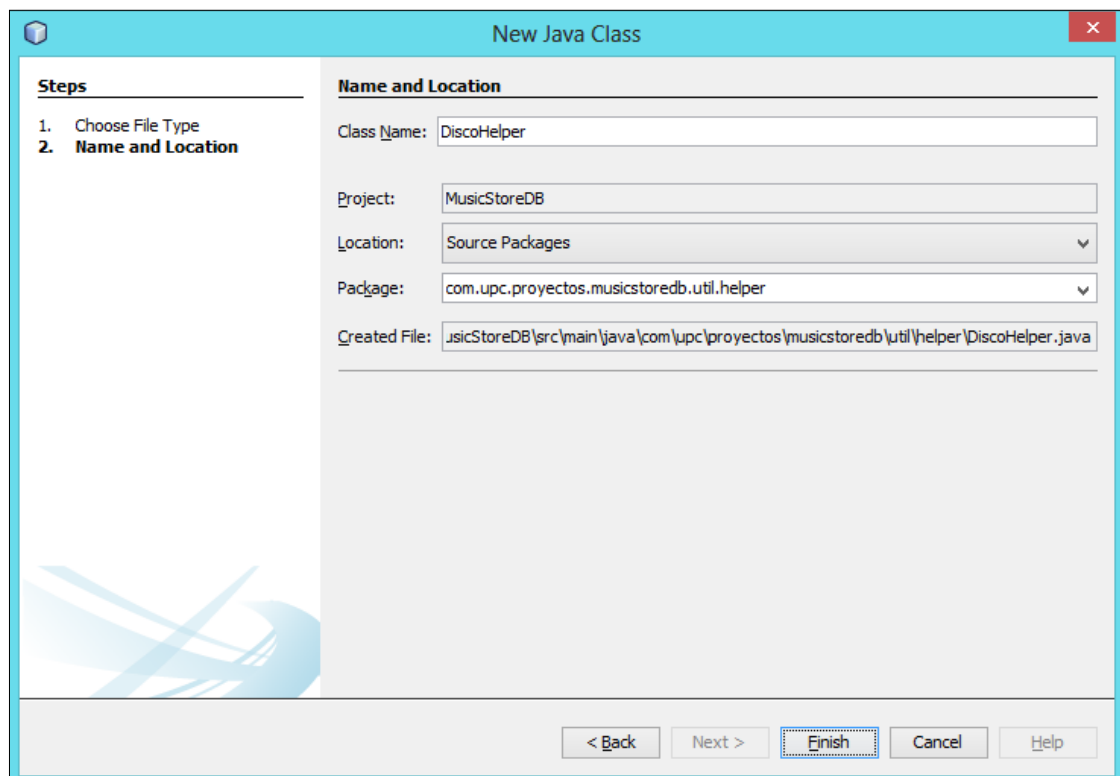
Clic derecho en nuestro proyecto y agregamos **HibernateUtil.java** y clic en Next.



Llamamos al archivo **HibernateUtil** y lo ponemos en el paquete **com.upc.proyectos.musicstoredb.util**. Esto nos creará la clase que usará hibernate.cfg para conectarse a la base de datos. Clic en Finish.



Creamos una clase que se llame **DiscoHelper** y la ponemos en el paquete **com.upc.proyectos.musicstoredb.util.helper**



La clase DiscoHelper se encargará de gestionar la conexión e implementará los métodos CRUD de la tabla Disco. Sin más detalles, ya que el objetivo es Web Services, la clase quedará de la siguiente manera:

```
package com.upc.proyectos.musicstoredb.util.helper;

import com.upc.proyectos.musicstoredb.entidades.Disco;
import com.upc.proyectos.musicstoredb.util.HibernateUtil;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class DiscoHelper {

    public DiscoHelper() {

    }

    private Session sesion;
    private Transaction tx;

    public Disco guardaDisco(Disco Disco) throws HibernateException {
        long id = 0;

        try {
            iniciaOperacion();
            id = (Long) sesion.save(Disco);
            tx.commit();
            Disco.setId(id);
        } catch (HibernateException he) {
            manejaExcepcion(he);
            throw he;
        } finally {
            sesion.close();
        }

        return Disco;
    }

    public Disco actualizaDisco(Disco Disco) throws HibernateException {
        try {
            iniciaOperacion();
            sesion.update(Disco);
            tx.commit();
        } catch (HibernateException he) {
            manejaExcepcion(he);
            throw he;
        } finally {
            sesion.close();
        }

        return Disco;
    }
}
```

```

    }

    public void eliminaDisco(Disco Disco) throws HibernateException {
        try {
            iniciaOperacion();
            sesion.delete(Disco);
            tx.commit();
        } catch (HibernateException he) {
            manejaExcepcion(he);
            throw he;
        } finally {
            sesion.close();
        }
    }

    public Disco obtenDisco(long idDisco) throws HibernateException {
        Disco Disco = null;
        try {
            iniciaOperacion();
            Disco = (Disco) sesion.get(Disco.class, idDisco);
        } finally {
            sesion.close();
        }

        return Disco;
    }

    public List<Disco> obtenListaDiscos() throws HibernateException {
        List<Disco> listaDiscos = null;

        try {
            iniciaOperacion();
            listaDiscos = sesion.createQuery("from Disco").list();
        } finally {
            sesion.close();
        }

        return listaDiscos;
    }

    private void iniciaOperacion() throws HibernateException {
        sesion = HibernateUtil.getSessionFactory().openSession();
        tx = sesion.beginTransaction();
    }

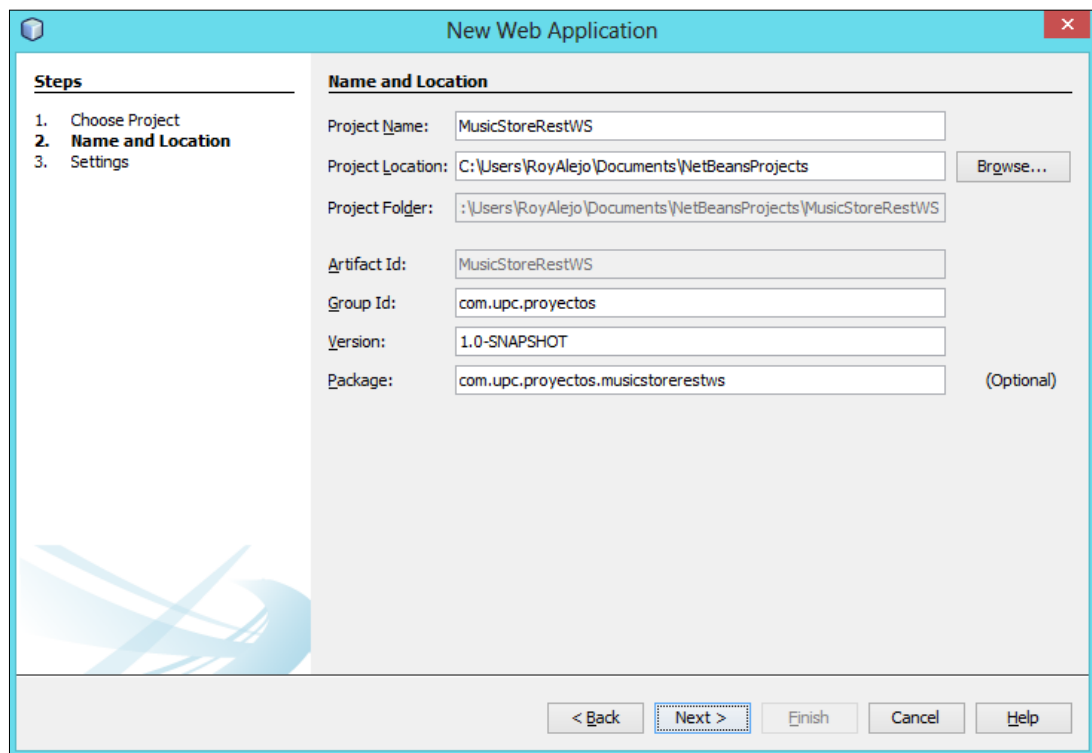
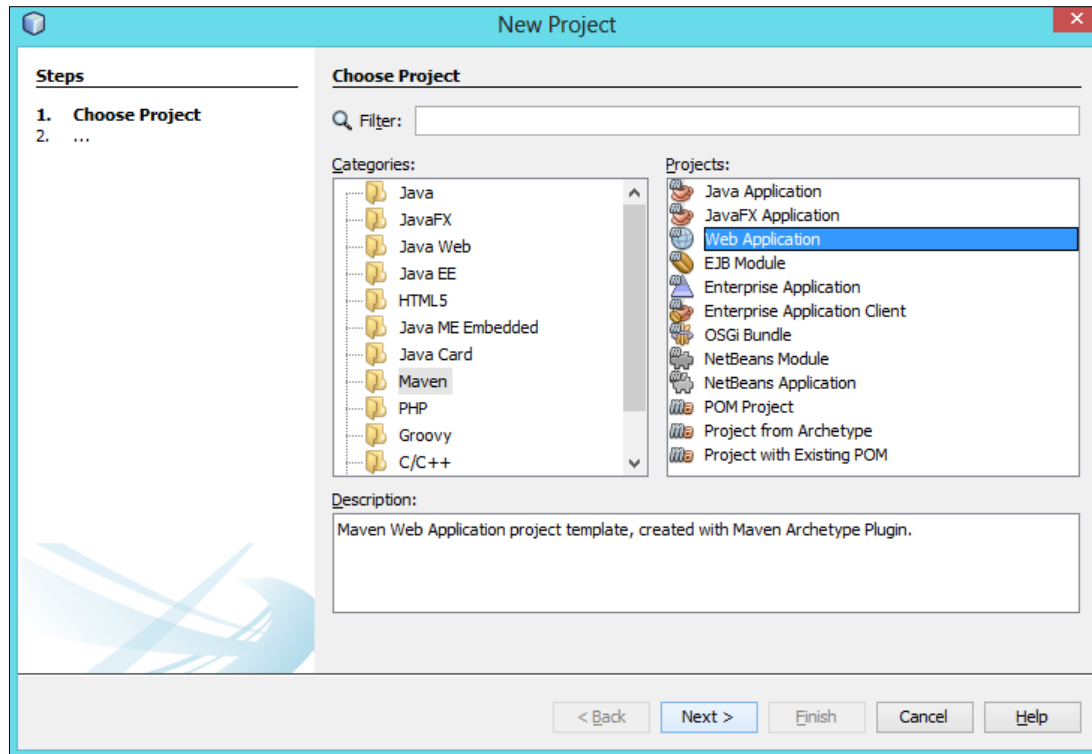
    private void manejaExcepcion(HibernateException he) throws HibernateException {
        tx.rollback();
        throw new HibernateException("Ocurrió un error en la capa de acceso a datos", he);
    }
}

```

Terminamos, ahora crearemos los proyectos que harán el CRUD de disco utilizando web services.

## Web Services – Rest

Creamos un proyecto Maven -> Web Application y lo nombramos **MusicStoreRestWS**

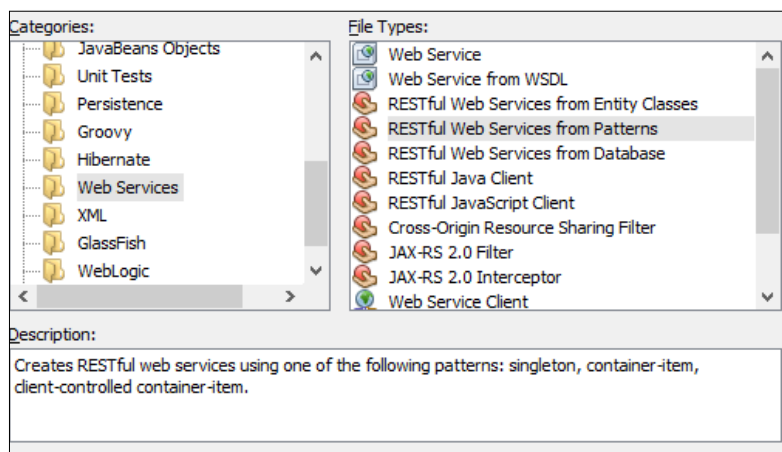


Clic Next y luego Finish.

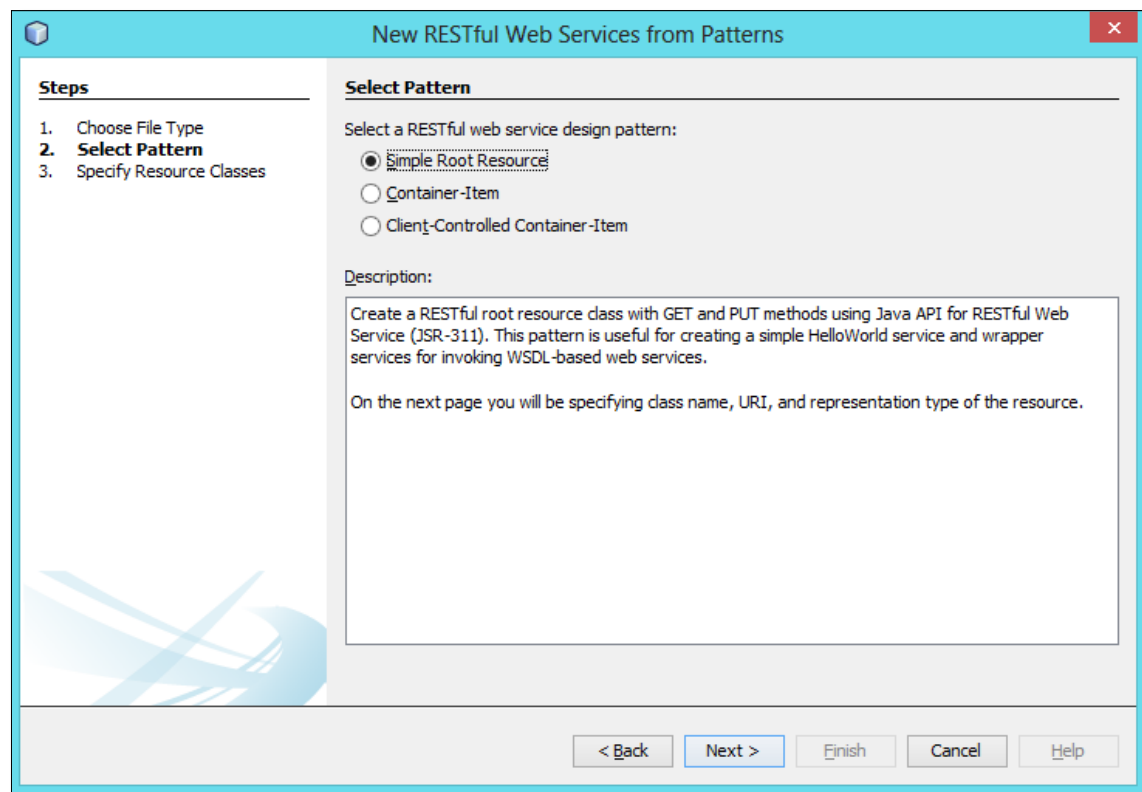
Nuestro proyecto tendrá las dependencias hibernate, jersey (para el estándar de los servicios), MySQL y al proyecto MusicStoreDB.

```
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet</artifactId>
  <version>2.22.2</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jersey.core</groupId>
  <artifactId>jersey-client</artifactId>
  <version>2.22.2</version>
</dependency>
<dependency>
  <groupId>com.upc.proyectos</groupId>
  <artifactId>MusicStoreDB</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.3.1.Final</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.23</version>
</dependency>
```

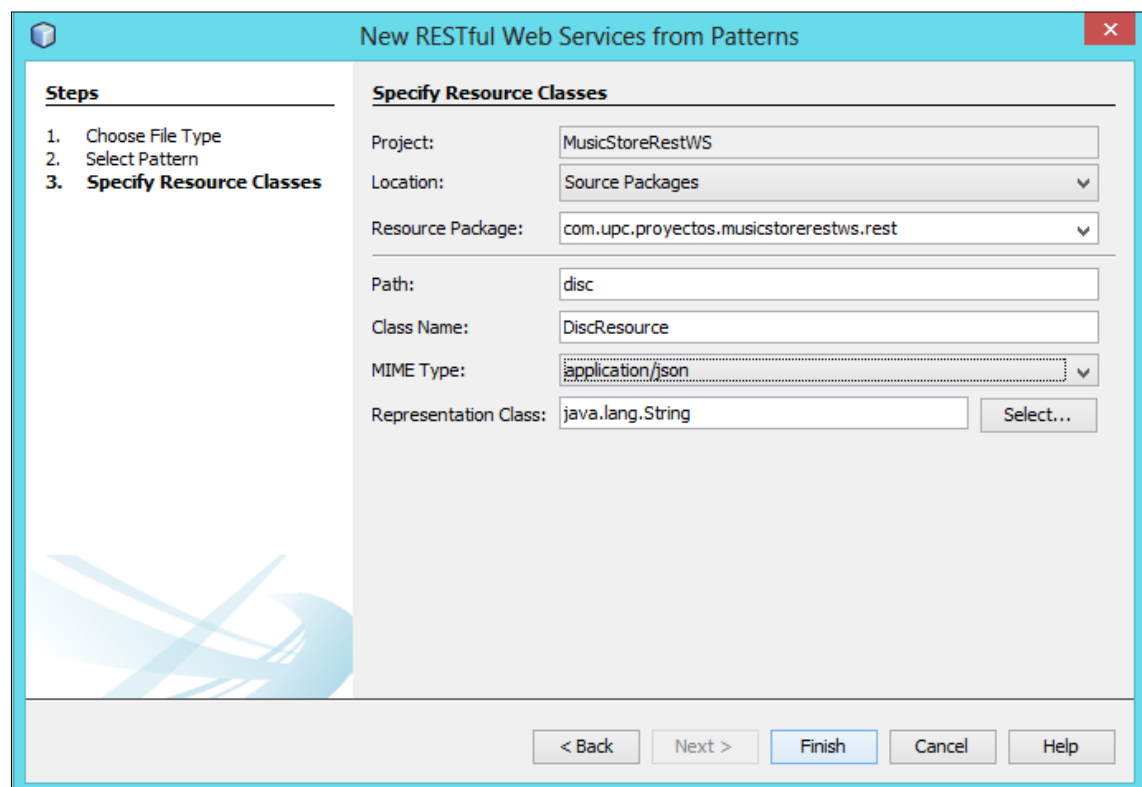
Clic derecho en nuestro proyecto y agregamos un archivo **RESTful Web Services from Patterns**.



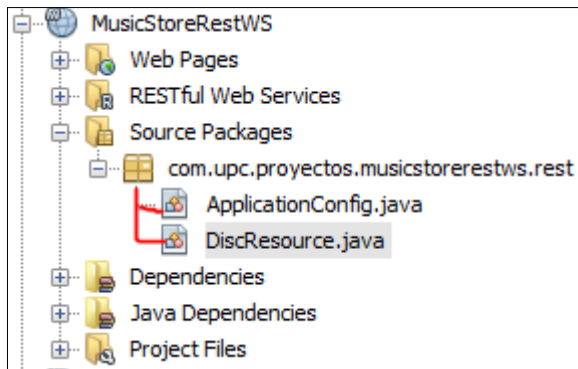
Seleccionamos **Simple Root Resource**. Clic en Next.



Paquete **com.upc.proyectos.musicstorerestdws.rest**, Path **disc** y Class Name **DiscResource**. Application type como **application/json**. Clic en Finish.



Esto nos creará 2 clases.



ApplicationConfig: esta clase no la modificaremos, sólo nos interesaría la anotación `ApplicationPath`, este valor nos indicará en que URL estará ubicado el servicio.

Ej: <http://localhost:8080/MusicStoreRestWS/webresources>

```
package com.upc.proyectos.musicstorere RestWS.rest;

import java.util.Set;
import javax.ws.rs.core.Application;

@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new java.util.HashSet<>();
        addRestResourceClasses(resources);
        return resources;
    }

    private void addRestResourceClasses(Set<Class<?>> resources) {
    }
}
```

DiscResource: acá implantaremos los servicios. Lo que podemos mencionar primero es que esta clase también tiene una anotación con la que se accederá al servicio, que en este caso será `Path="disc"`

Ej: <http://localhost:8080/MusicStoreRestWS/webresources/disc>



```

package com.upc.proyectos.musicstorerestdws.rest;

import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PUT;

@Path("disc")
public class DiscResource {

    @Context
    private UriInfo context;

    public DiscResource() {
    }

    @GET
    @Produces("application/json")
    public String getJson() {
        //TODO return proper representation object
        throw new UnsupportedOperationException();
    }

    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }
}

```

### Implementemos.

Borramos las funciones getJson y putJson con sus anotaciones y definimos un atributo DiscoHelper y lo iniciamos en el constructor de la clase. Quedaría de la siguiente manera:

```

package com.upc.proyectos.musicstorerestdws.rest;

import com.upc.proyectos.musicstoredb.util.helper.DiscoHelper;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PUT;

@Path("disc")

```

```

public class DiscResource {

    private DiscoHelper helper;

    @Context
    private UriInfo context;

    public DiscResource() {
        helper = new DiscoHelper();
    }
}

```

### Listar

Cada servicio será una función de la clase DiscResource. Para empezar, haremos el servicio de Listar los discos. Para ello crearemos una función listarDiscos que no recibirá parámetros y retornará la lista de discos. Agregaremos las anotaciones GET y Produces(json), quedará así:

```

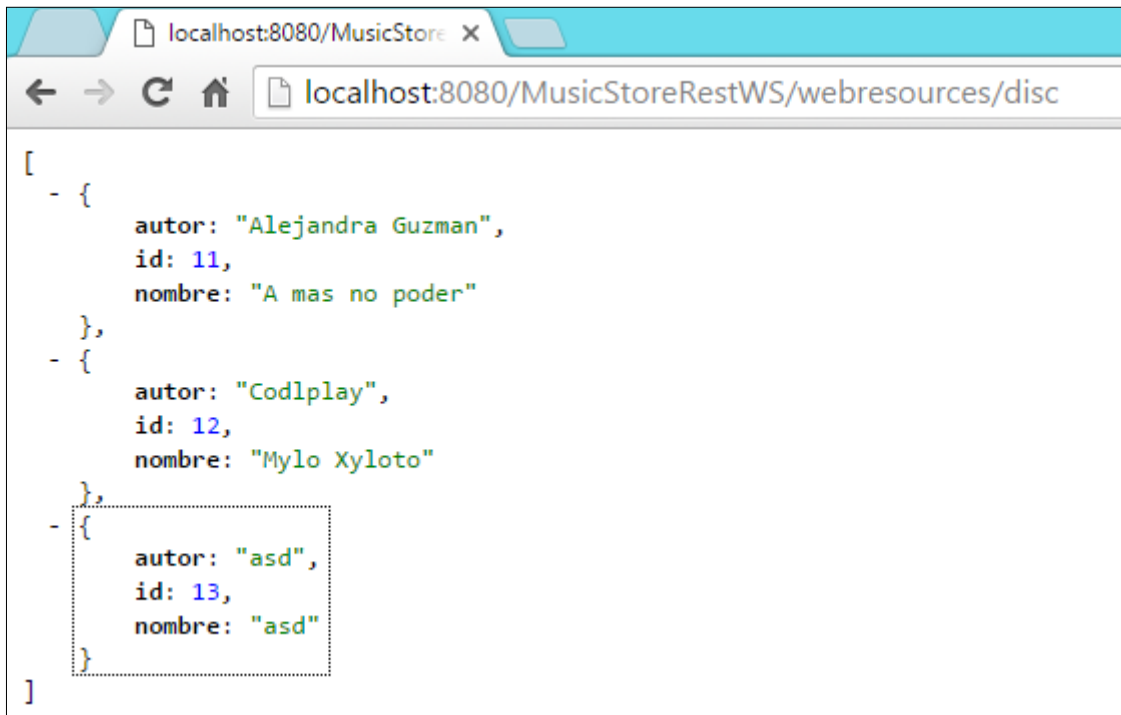
@GET
@Produces(MediaType.APPLICATION_JSON)
public ArrayList<Disco> listarDiscos(){
    return new ArrayList(helper.obtenListaDiscos());
}

```

La anotación GET indica que se accederá mediante GET. Produces indica que el servicio devolverá un archivo json como respuesta.

Desplegamos el proyecto e ingresamos a la siguiente ruta:

<http://localhost:8080/MusicStoreRestWS/webresources/disc>

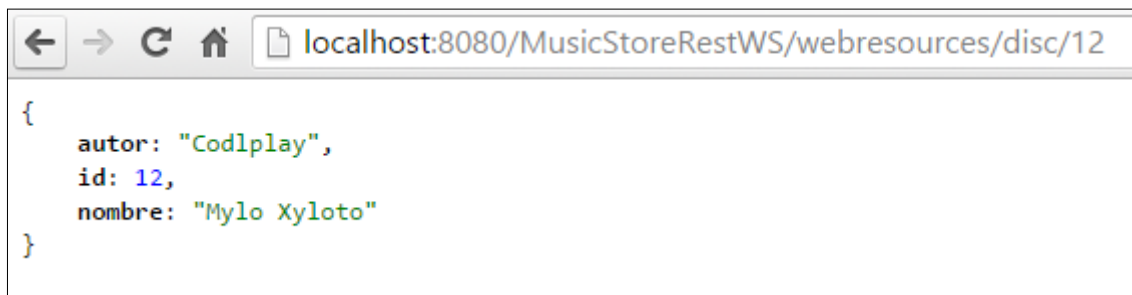


## Obtener

Ahora implementaremos obtener 1 disco por su id. Creamos una función de la siguiente manera y probamos:

```
@Path("/{id}")
@GET
@Produces(MediaType.APPLICATION_JSON)
public Disco obtenerDisco(@PathParam("id") long idDisco){
    return helper.obtenDisco(idDisco);
}
```

Accedemos a la url <http://localhost:8080/MusicStoreRestWS/webresources/disc/12>



Ya sabemos para qué son las anotaciones GET y Produces. Lo nuevo acá es Path y PathParam.

**Path:** indicamos cuál será la ruta con la que se accederá. Si ponemos un valor entre llaves será una variable que será seteada con PathParam.

Ejemplos:

Path("/listarObjetos")

<http://localhost:8080/MusicStoreRestWS/webresources/disc/listarObjetos>

Path("/servicioAdicional")

<http://localhost:8080/MusicStoreRestWS/webresources/disc/servicioAdicional>

Path("/{id}")

<http://localhost:8080/MusicStoreRestWS/webresources/disc/7>

Path("/buscarPorAutor/{nombre}")

<http://localhost:8080/MusicStoreRestWS/webresources/disc/buscarPorAutor/Coldplay>

**Pasar Parámetros al servicio:** hay varias formas de pasar parámetros a un servicio, en esta ocasión hemos visto pasarle el **id** del disco como PathParam, el valor que se pone indica el nombre del parámetro para invocar el servicio, como vemos no es necesario que sea el mismo que el de la función. Estas son las otras formas de parámetros:

**@PathParam**

<http://localhost:8080/MusicStoreRestWS/webresources/disc/Marcian/H2K>

**@QueryParam**

[http://localhost:8080/MusicStoreRestWS/webresources/disc?titulo="Marcian"&autor="H2K"](http://localhost:8080/MusicStoreRestWS/webresources/disc?titulo=)

## @MatrixParam

[http://localhost:8080/MusicStoreRestWS/webresources/disc?titulo =\"Marcian\";autor= \"H2K\"](http://localhost:8080/MusicStoreRestWS/webresources/disc?titulo =\)

## @FormParam

Si tenemos un formulario con 2 campos inputs y 1 botón submit, permite al cliente entrar a estos detalles y enviarlos al servicio RESTful. Entonces el servicio rest extraerá estos campos usando la anotación @FormParam.

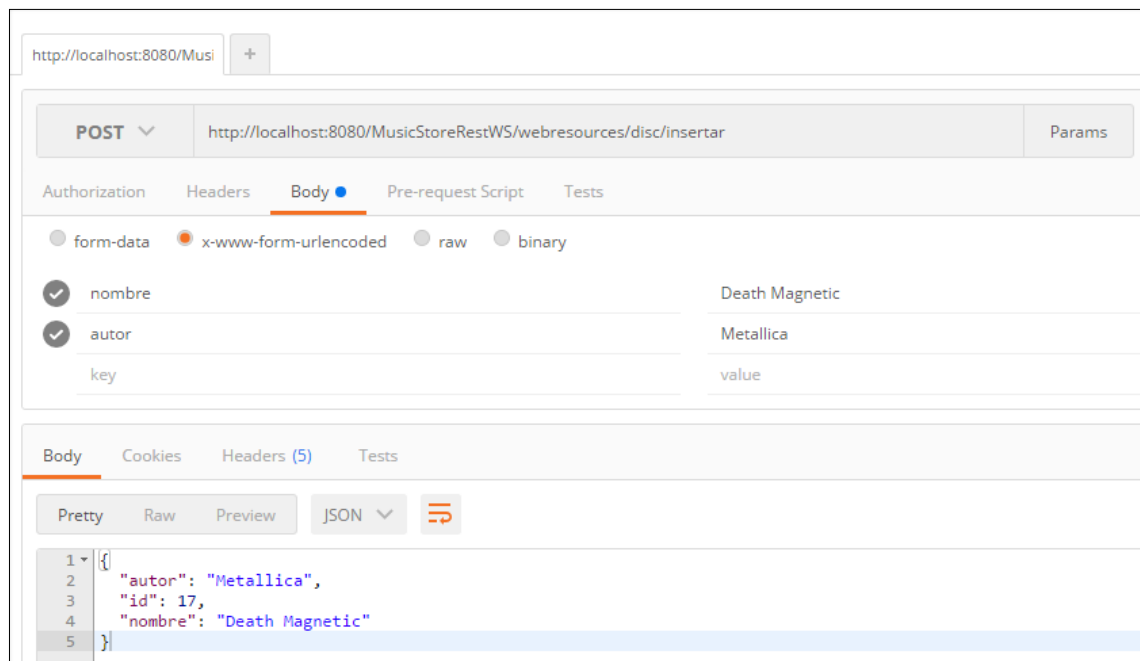
### Insertar

Ahora crearemos las funciones de INSERTAR. Este método será:

- POST
- Path: /insertar
- QueryParam: nombre y autor
- Consumes: application/x-www-form-urlencoded (indica que tipo de recurso será aceptado por el servicio)
- Produces: json

```
@Path("/insertar")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Disco insertarDisco(@FormParam("nombre") String nombre,
    @FormParam("autor") String autor){
    Disco disco = new Disco(nombre, autor);
    return helper.guardaDisco(disco);
}
```

Para probar el servicio utilizaré la herramienta Postman:



Si consumimos el servicio listar (<http://localhost:8080/MusicStoreRestWS/webresources/disc>) nos mostrará lo que insertamos.

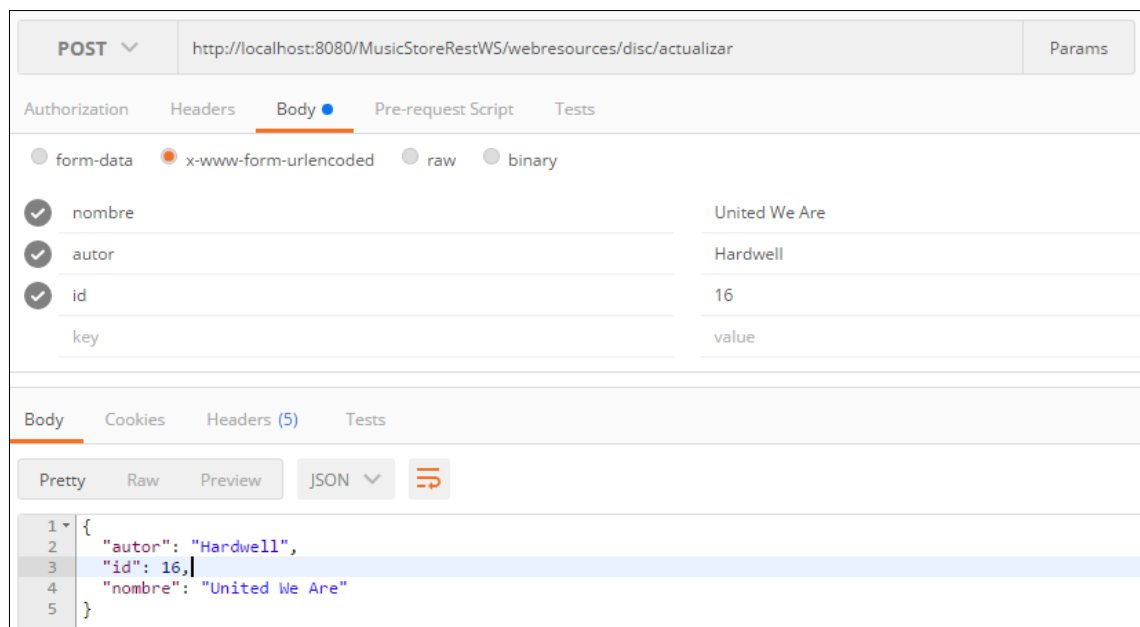
## Actualizar

Para Actualizar será lo mismo, a diferencia que cambiaremos el path y añadiremos el id de la entidad que queremos actualizar. Este método será:

- POST
- Path: /actualizar
- QueryParam: id, nombre y autor
- Consumes: application/x-www-form-urlencoded
- Produces: json

```
@Path("/actualizar")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Disco actualizarDisco(@FormParam("id") long id, @FormParam("nombre") String
nombre, @FormParam("autor") String autor){
    Disco disco = helper.obtenDisco(id);
    disco.setNombre(nombre);
    disco.setAutor(autor);
    return helper.actualizaDisco(disco);
}
```

Probamos:



## Eliminar

Para Eliminar seguiremos la misma dinámica. En esta ocasión sólo se enviará el id a eliminar y se nos devolverá un mensaje.

- GET
- Path: /eliminar/{id}
- PathParam: id
- Produces: json

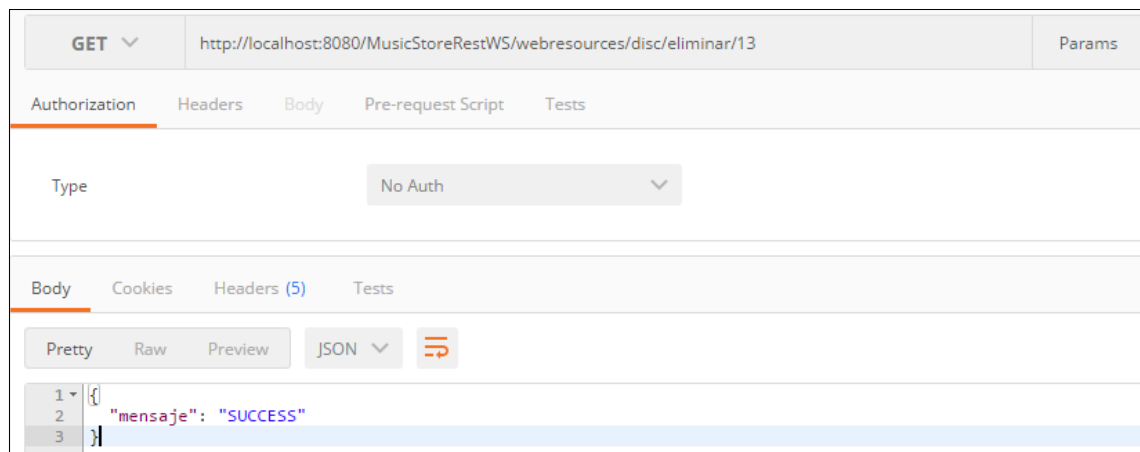
```
@Path("/eliminar/{id}")
```

```

@GET
@Produces(MediaType.APPLICATION_JSON)
public String eliminarDisco(@PathParam("id") long id) {
    Disco disco = helper.obtenDisco(id);
    if (disco == null) {
        return "{\"mensaje\":\"FAIL\"}";
    } else {
        helper.eliminaDisco(disco);
        return "{\"mensaje\":\"SUCCESS\"}";
    }
}

```

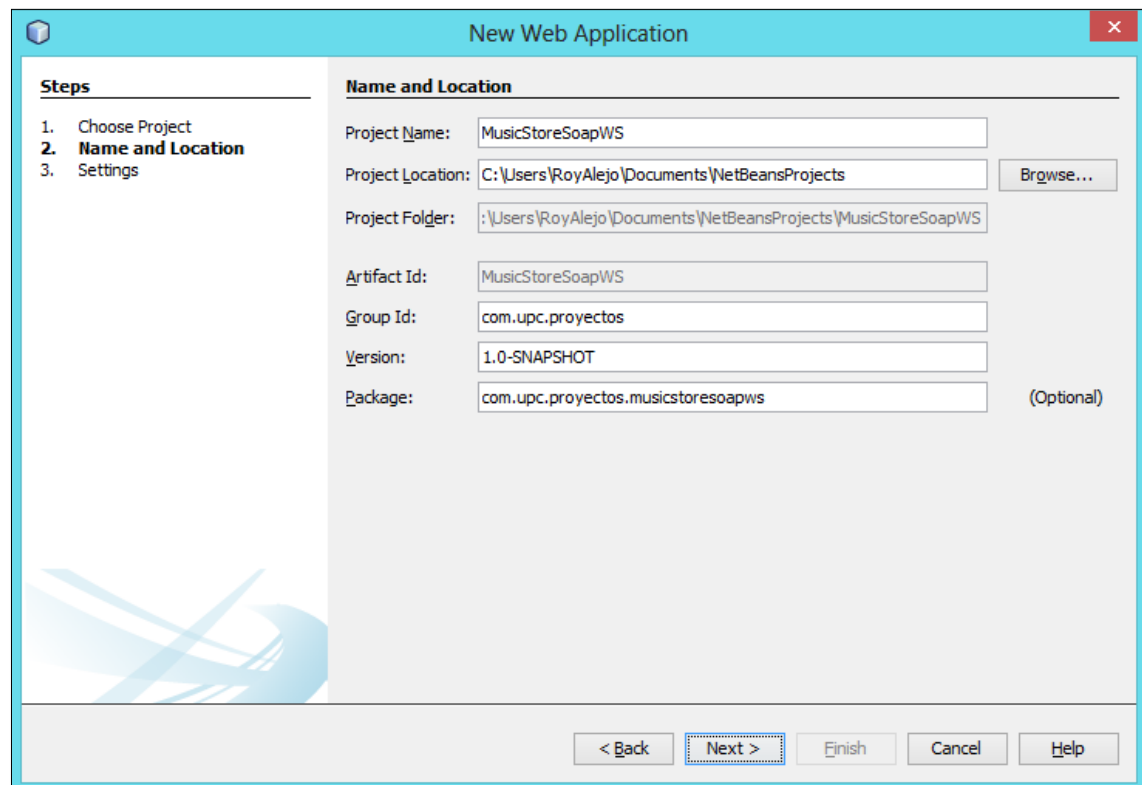
Probamos:



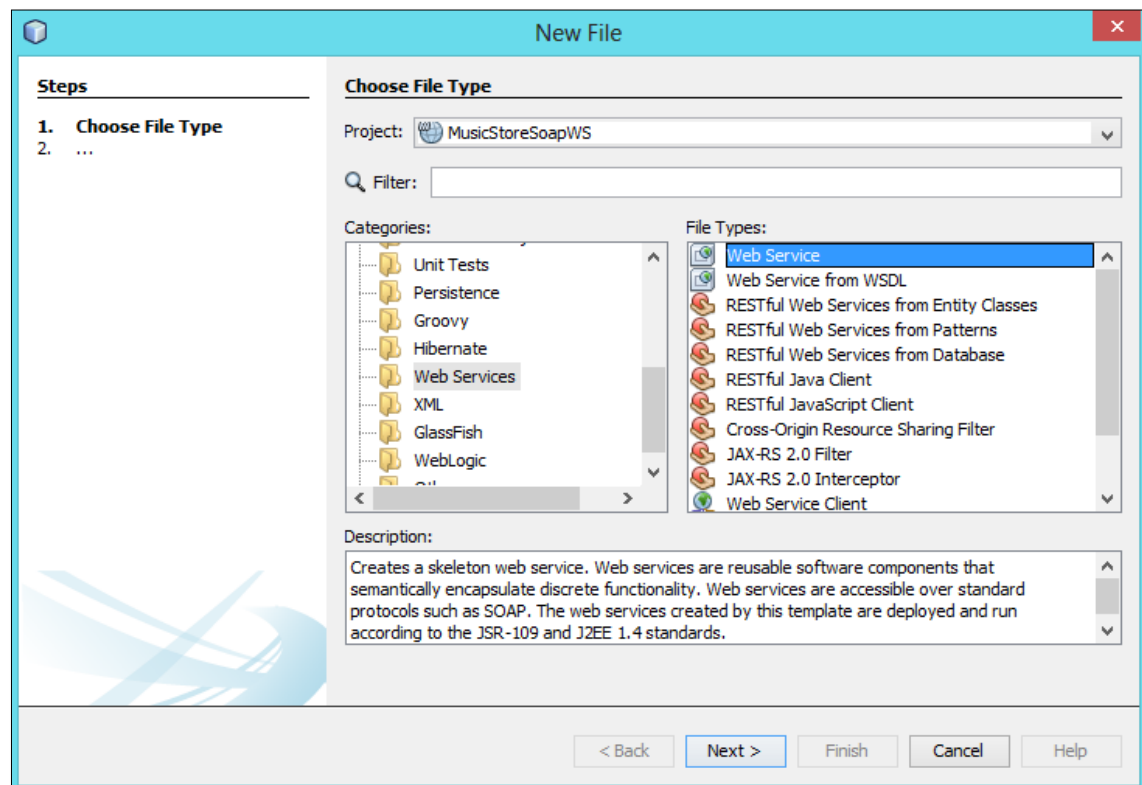
Listo, hemos revisado como crear servicios REST con JAVA.

## Web Services – SOAP

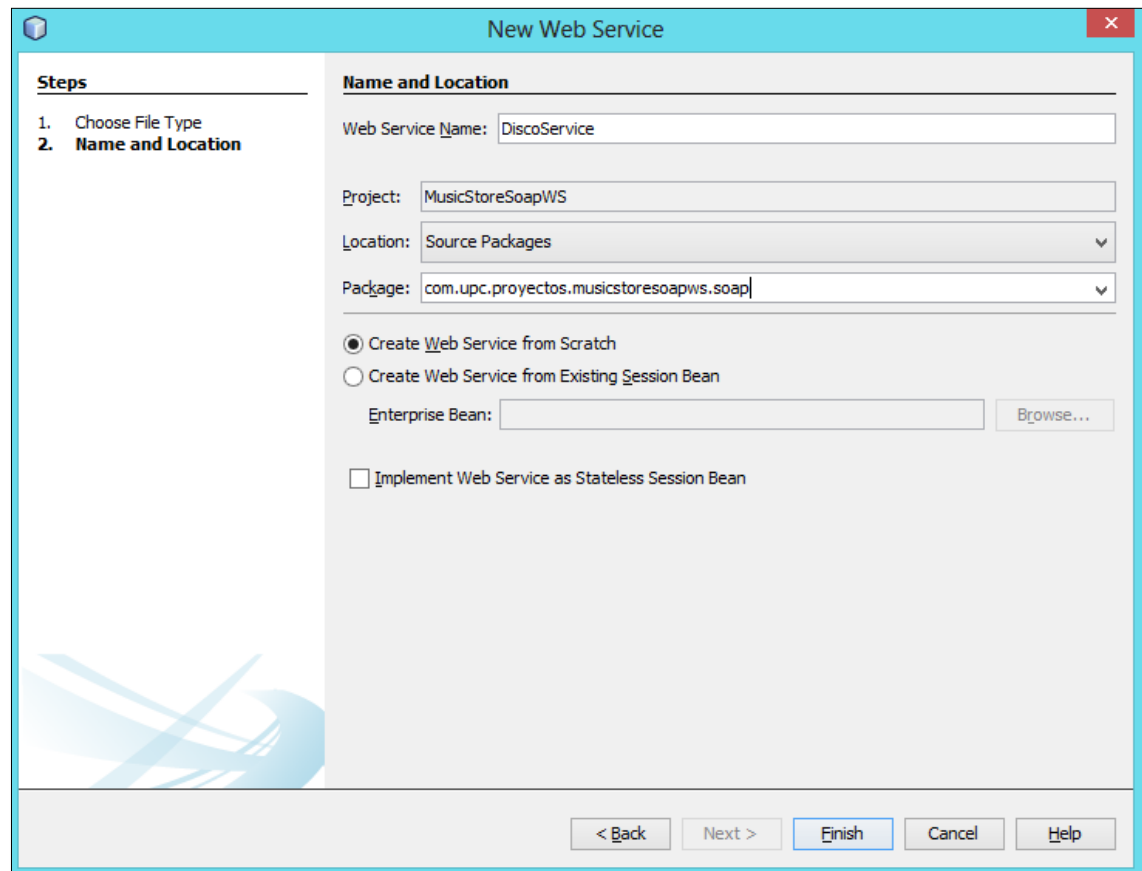
Creamos un proyecto Maven -> Web Application y lo nombramos **MusicStoreSoapWS**



Clic derecho sobre nuestro proyecto y creamos un **Web Service**



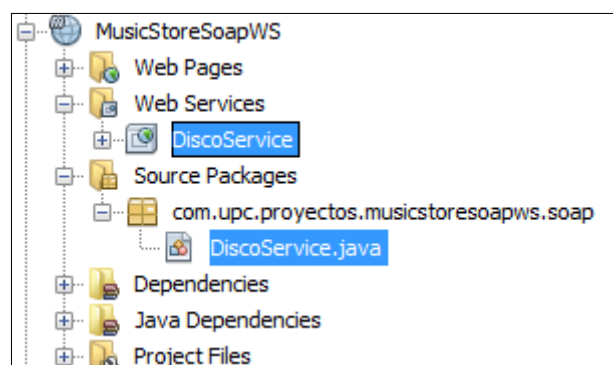
Lo llamamos **DiscoService**, lo ponemos en el paquete **com.upc.proyectos.musicstoresoapws.soap** y clic en Finish.



Agregaremos la dependencia de nuestro proyecto MusicStoreDB y webservices-rt(aunque esta dependencia se agrega automáticamente al crear el servicio web:

```
<dependency>
  <groupId>com.upc.demohibernate</groupId>
  <artifactId>MusicStoreDB</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

Notemos que la estructura del proyecto es la siguiente:



CRUD completo

Vayamos a la clase DiscoService, borramos todo y pegamos lo siguiente:



```

package com.upc.proyectos.musicstoresoapws.soap;

import com.upc.proyectos.musicstoredb.entidades.Disco;
import com.upc.proyectos.musicstoredb.util.helper.DiscoHelper;
import java.util.List;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "DiscoService")
public class DiscoService {

    @WebMethod(operationName = "listar")
    public List<Disco> listarDiscos() {
        return new DiscoHelper().obtenListaDiscos();
    }

    @WebMethod(operationName = "obtener")
    public Disco obtenerDisco(@WebParam(name = "id") long id) {
        return new DiscoHelper().obtenDisco(id);
    }

    @WebMethod(operationName = "insertar")
    public Disco insertarDisco(@WebParam(name = "titulo") String titulo, @WebParam(name = "autor") String autor) {
        Disco disco = new Disco(titulo, autor);
        return new DiscoHelper().guardaDisco(disco);
    }

    @WebMethod(operationName = "actualizar")
    public Disco actualizarDisco(@WebParam(name = "id") long id, @WebParam(name = "nombre") String nombre, @WebParam(name = "autor") String autor) {
        DiscoHelper discoHelper = new DiscoHelper();
        Disco disco = discoHelper.obtenDisco(id);
        disco.setNombre(nombre);
        disco.setAutor(autor);
        return discoHelper.actualizaDisco(disco);
    }

    @WebMethod(operationName = "borrar")
    public String borrarDisco(@WebParam(name = "id") long id) {
        DiscoHelper discoHelper = new DiscoHelper();
        Disco disco = discoHelper.obtenDisco(id);
        if (disco != null) {
            discoHelper.eliminaDisco(disco);
        }
        return "Success!";
    }
}

```

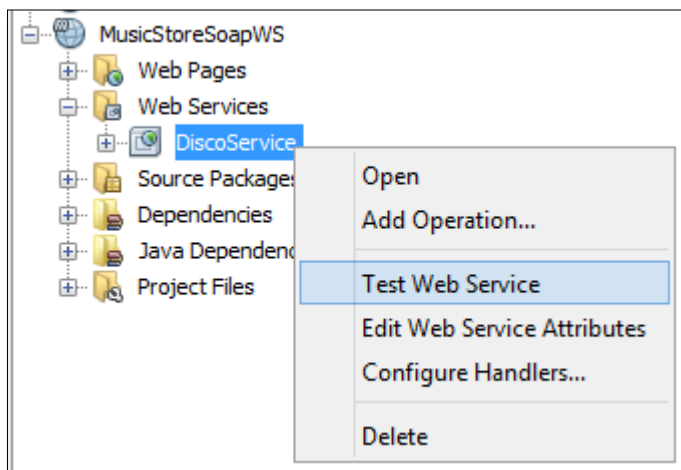
Observemos:

- La clase tiene la anotación **@WebService(serviceName = "DiscoService")**, esto indica el nombre del servicio.
- Cada función tiene la anotación **@WebMethod(operationName = "listar")**, esto indica el nombre de la operación del servicio.
- Cada parámetro de cada función se le antepone la anotación **@WebParam(name = "id")**, esto indica al servicio el nombre del parámetro de la operación del servicio.
- No necesariamente el nombre del servicio es el mismo que el de la clase.
- No necesariamente el nombre del método es el mismo que el de la función.
- No necesariamente el nombre del parámetro es el mismo que el parámetro de la función.

El CRUD de la tabla Disco con servicios SOAP está listo.

### Probando

Seleccionamos la opción **Test Web Service** como indica la imagen:



The image shows a web browser window titled 'DiscoService Web Service'. The address bar shows 'localhost:8080/MusicStoreSoapWS/DiscoService?Tester'. The page content is titled 'DiscoService Web Service Tester'. Below the title, there is a description: 'This form will allow you to test your web service implementation (WSDL File)'. A note says: 'To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.' Under the heading 'Methods:', there are five method signatures, each with a button and input fields:
 

- `public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.actualizar(long java.lang.String java.lang.String)` with a button labeled 'actualizar' and two input fields.
- `public abstract java.lang.String com.upc.proyectos.musicstoresoapws.soap.DiscoService.borrar(long)` with a button labeled 'borrar' and one input field.
- `public abstract java.util.List com.upc.proyectos.musicstoresoapws.soap.DiscoService.listar()` with a button labeled 'listar' and no input fields.
- `public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.obtener(long)` with a button labeled 'obtener' and one input field.
- `public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.insertar(java.lang.String java.lang.String)` with a button labeled 'insertar' and two input fields.

Comenzamos a probar nuestros servicios.

- Listar

Hacemos clic en Listar.

```
public abstract java.util.List com.upc.proyectos.musicstoresoapws.soap.DiscoService.listar()
listar ()
```

Podemos ver el resultado:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:listarResponse xmlns:ns2="http://soap.musicstoresoapws.proyectos.upc.com/">
      <return>
        <autor>Metallica</autor>
        <id>1</id>
        <titulo>a</titulo>
      </return>
      <return>
        <autor>Coldplay</autor>
        <id>2</id>
        <titulo>a</titulo>
      </return>
      <return>
        <autor>Muse</autor>
        <id>3</id>
        <titulo>a</titulo>
      </return>
      <return>
        <autor>Michael</autor>
        <id>8</id>
        <titulo>a</titulo>
      </return>
      <return>
        <autor>Mylo Xyloto</autor>
        <id>10</id>
        <titulo>a</titulo>
      </return>
    </ns2:listarResponse>
  </S:Body>
</S:Envelope>
```

- Insertar

Llenamos el formulario de Insertar y clic en Insertar

```
public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.insertar(java.lang.String,java.lang.String)
insertar (testNombre, testAutor)
```

Vemos el resultado:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:insertarResponse xmlns:ns2="http://soap.musicstoresoapws.proyectos.upc.com/">
      <return>
        <autor>testAutor</autor>
        <id>11</id>
        <titulo>testNombre</titulo>
      </return>
    </ns2:insertarResponse>
  </S:Body>
</S:Envelope>
```

- Actualizar

Llenamos el formulario de Actualizar y listo.

```
public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.actualizar(long java.lang.String java.lang.String)
actualizar (11 Before the Storm Darude)
```

Vemos el resultado:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:actualizarResponse xmlns:ns2="http://soap.musicstoresoapws.proyectos.upc.com/">
      <return>
        <autor>Darude</autor>
        <id>11</id>
        <titulo>Darude</titulo>
      </return>
    </ns2:actualizarResponse>
  </S:Body>
</S:Envelope>
```

- Obtener

Insertamos el id de lo que acabamos de actualizar:

```
public abstract com.upc.proyectos.musicstoresoapws.soap.Disco com.upc.proyectos.musicstoresoapws.soap.DiscoService.obtener(long)
obtener (11)
```

Resultado:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:obtenerResponse xmlns:ns2="http://soap.musicstoresoapws.proyectos.upc.com/">
      <return>
        <autor>Darude</autor>
        <id>11</id>
        <titulo>Darude</titulo>
      </return>
    </ns2:obtenerResponse>
  </S:Body>
</S:Envelope>
```

- Eliminar

Ponemos el id de lo que acabamos de obtener y listo.