Evaluación.
Frameworks para Java EE

Telefónica
EDUCACIÓN DIGITAL

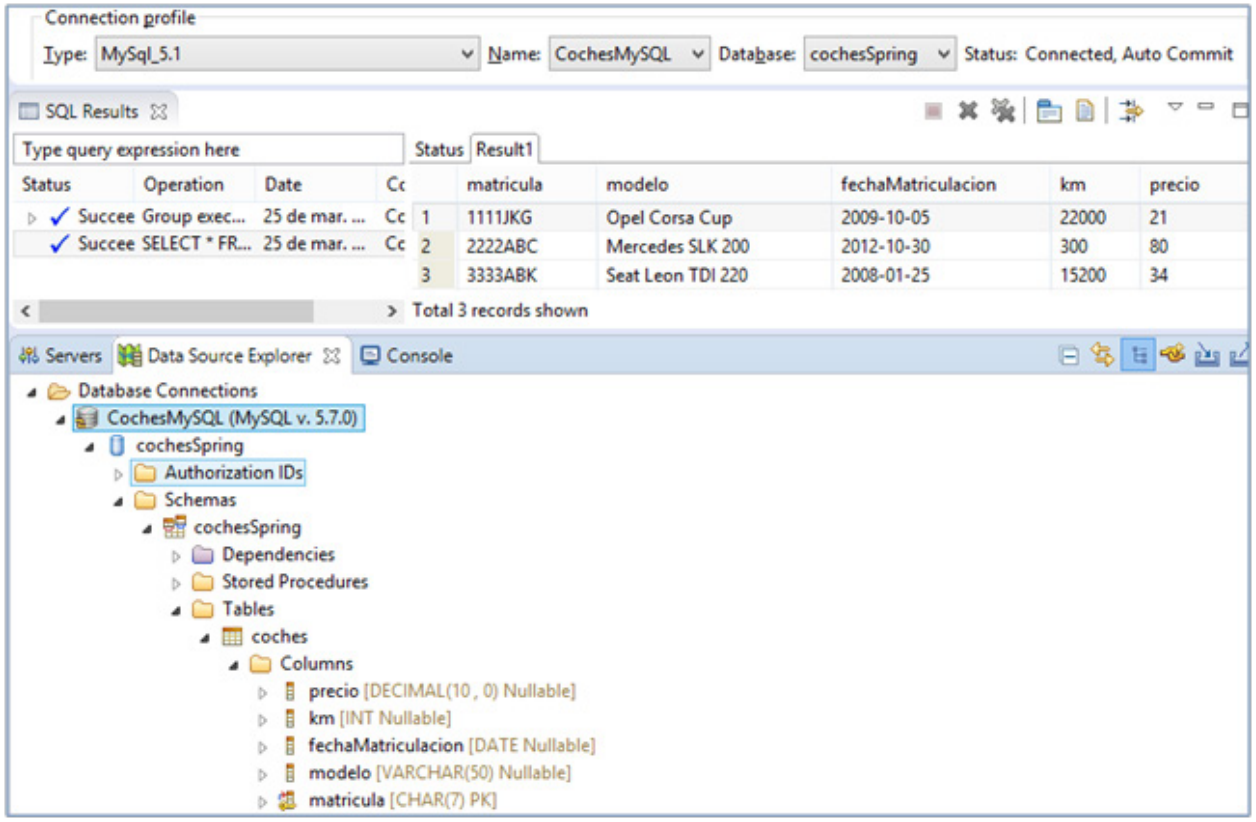
Caso práctico

Se trata de hacer una aplicación web con Spring-MVC y acceso a datos con JdbcTemplate.

La aplicación trabaja con una tabla "coches" de la base de datos "cochesSpring" de MySQL, cuya estructura es generada con la sentencia SQL siguiente.

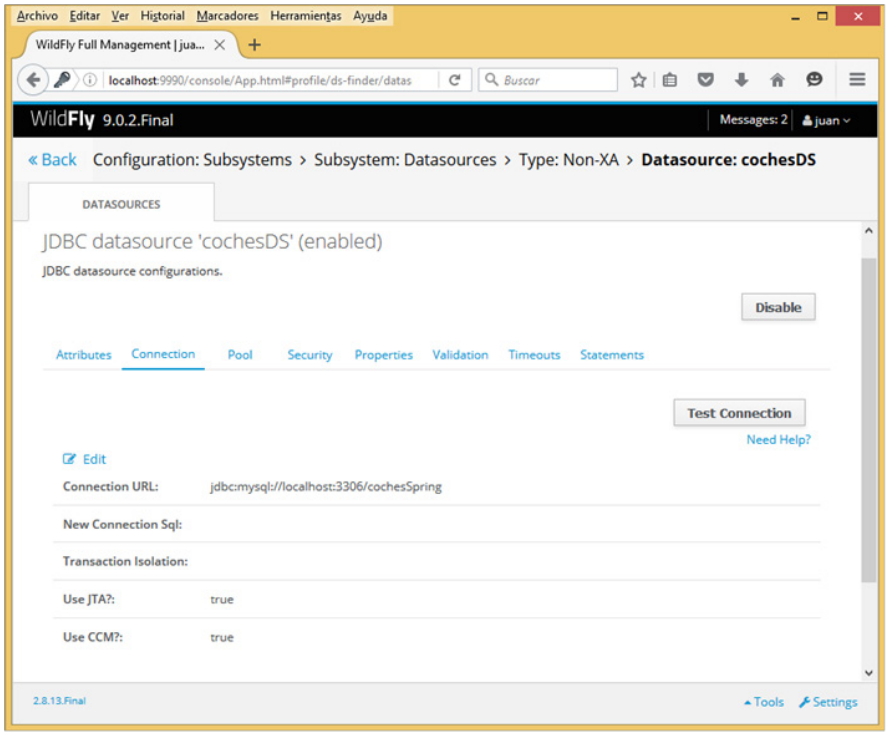
```
CREATE TABLE coches (
  matricula char(7) NOT NULL PRIMARY KEY,
  modelo varchar(50) default NULL,
  fechaMatriculacion date default NULL,
  km int default NULL,
  precio decimal(10,0) default NULL
)
```

La conexión, la base de datos, la tabla, los campos y un ejemplo de registros almacenados, vistos utilizando las utilidades de Spring "Data source Explorer" y "Sql Scrap Book", es como muestra la imagen siguiente.



Base de datos, tabla y registros

La conexión a la base de datos se hará utilizando un recurso "datasource" localizado con JDNI, creado con WildFly, tal y como muestra la imagen siguiente.



Recurso JNDI para datasource de la conexión a BBDD

La aplicación presentara tres páginas HTML, generadas por las siguientes páginas jsp:

- "listar.jsp", será la que aparezca cuando arranque la aplicación, presentara los registros de la tabla de la base de datos.
- En esta lista cada registro tendrá un hipervínculo que hará una petición al jsp "editar.jsp", que permitirá cambiar cualquier dato del coche, excepto la matricula.
- "crear.jsp", será la encargada de presentar un formulario para introducir los datos de un nuevo coche, y recibirá la petición de un hipervínculo de "listar.jsp"

La aplicación tendrá un "controller" anotado con @Controller, tendrá:

- Una referencia a la interfaz del modelo de negocio inyectada.
- Las funciones para listar, editar y crear que invocarán a las correspondientes funciones del objeto que implemente la interfaz DAO.

De lo comentado para "controller", se deduce que se tendrán que implementar, una interfaz y la clase que la implementa para la lógica del negocio y otra interfaz y su clase que la implementa para el patrón DAO.

- **Interfaz para el negocio y clase que lo implementa** con listar, editar y crear. La clase que implementa el negocio estará anotada con @Service y tendrá inyectado el bean de la clase que implementa la interfaz DAO.
- **Interfaz para el DAO y la clase que lo implementa**, Esta clase tendrá como datos:
  - **JdbcTemplate template**
  - **DataSource ds**

En una **clase interna** implementara el **RowMapper** para la clase entidad **Coche**.

Además implementará todas las funciones necesarias para dar servicio al objeto de negocio que le pida servicio. En estas funciones se utilizarán las funcionalidades de la plantilla **JdbcTemplate**, tales como query, queryForObject y update.

La información de cada registro para visualizarlo, insertarlo o modificarlo estará encapsulada en la clase entidad Coche, que tendrá un dato para campo de la tabla, con sus correspondientes funciones getter y setter.

La imagen siguiente muestra la relación entre las diferentes clases e interfaces de la lógica del negocio, dao, la entidad, el controller, el ServletDispatcher y las tres páginas jsp.

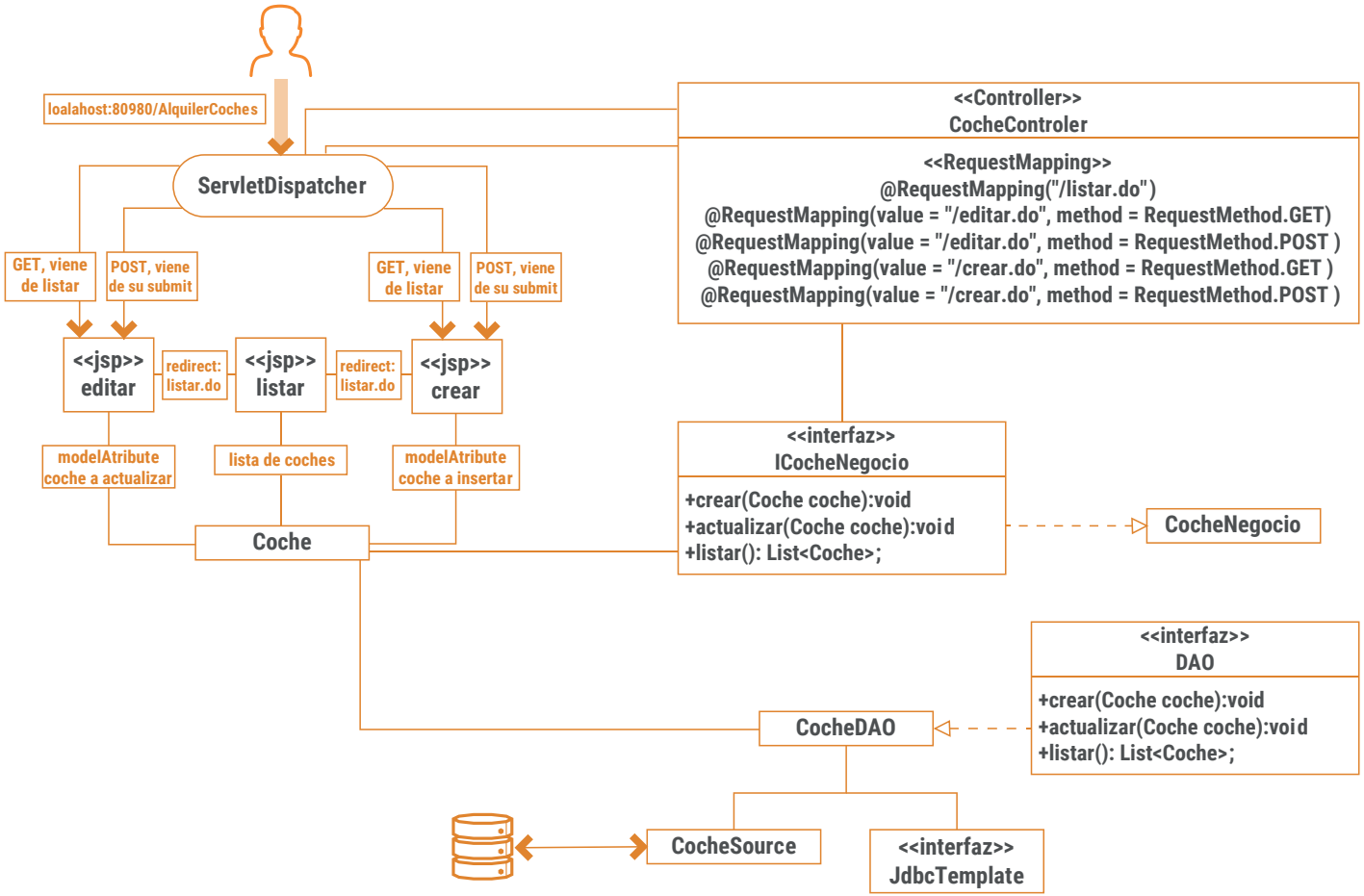


Diagrama con la relación de todos los elementos que intervienen en a aplicación

Como ayuda se indica el código que puede tener el fichero web.xml

```
<welcome-file-list>
        <welcome-file>listar.do</welcome-file>
</welcome-file-list>

<!-- beans de las capas de negocio y datos -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/config/springbeans.xml</param-value>
</context-param>

<listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>

<!-- MVC -->
<servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
                /WEB-INF/config/dispatcher-servlet.xml
                </param-value>
        </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

También como ayuda se muestra el contenido del fichero "springbeans.xml"

```
<context:component-scan base-package="com.juan"/>
<jee:jndi-lookup id="miDataSource" jndi-name="java/cochesDS" resource-
ref="true"/>
```

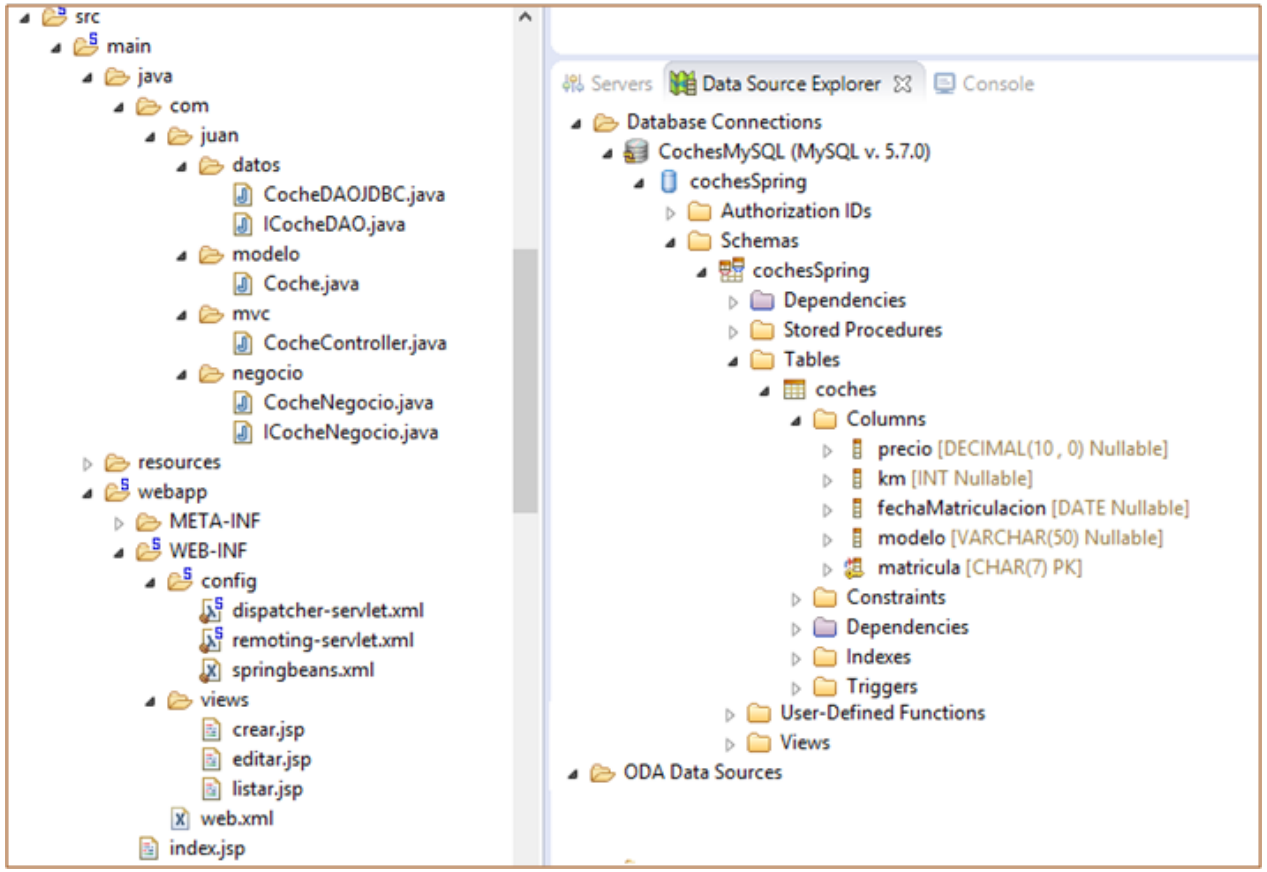
Para poder implementar correctamente la clase mapeada con @ Controller, se muestra el código de las dos funciones mapeadas con @RequestMapping que intervienen en la edición de un coche (cambios en sus datos.)

```
@RequestMapping(value = "/editar.do",
                        method = RequestMethod.GET)
public String prepararEditar(Model modelo,
                        @RequestParam(value = "matricula") String matricula)
{
                                                modelo.addAttribute

(cochenegocio.obtener(matricula));
                        return "editar";
}

@RequestMapping(value = "/editar.do",
                        method = RequestMethod.POST)
public String procesarEditar(Coche coche,
                        BindingResult result, Model modelo) {
                if (result.hasErrors()) {
                        return "editar";
                } else {
                        cochenegocio.actualizar(coche);
                        return "redirect:listar.do";
                }
}
```

La estructura del proyecto en Eclipse es como la que muestra la imagen siguiente.



Estructura del proyecto en Eclipse

Telefónica

EDUCACIÓN DIGITAL