

## Contents

<b>Tema 02: Conectividad con Bases de Datos Relacionales</b>	<b>2</b>
Índice de contenido . . . . .	3
El problema . . . . .	3
Preparando el entorno . . . . .	4
Clonando el proyecto . . . . .	4
Creando los contenedores para la base de datos . . . . .	4
<b>Preparando el entorno</b>	<b>5</b>
Creamos la BBDD . . . . .	6
<b>Creación del proyecto en modo interactivo (MAVEN)</b>	<b>10</b>
Dependencias Maven . . . . .	10
MySQL . . . . .	10
Soporte J2EE (Servlets) . . . . .	10
Soporte para JSST (para JSP) . . . . .	11
Soporte JSON . . . . .	11
Insertando el servidor Tomcat en Maven . . . . .	11
<b>Servlets</b>	<b>12</b>
Ciclo de vida . . . . .	12
Inicializar el servlet . . . . .	12
Interactuar con los clientes . . . . .	12
Destruir el servlet . . . . .	12
¿Cómo funciona un servlet? . . . . .	13
Preparando el entorno . . . . .	14
Creando el proyecto . . . . .	15
Selección del contenedor(servidor) J2EE. . . . .	16
Descarga e instalación de Tomcat. . . . .	17
Introducimos en NetBeans el usuario y contraseña de Tomcat . .	19
Estructura de nuestro proyecto Web con Servlets . . . . .	20
Añadiendo el primer servlet . . . . .	20
<b>Conexión a la base de datos</b>	<b>22</b>
Creando los Plain Old Java Objects . . . . .	22
Conectando a la base de datos cargando la configuración vía JNDI . .	31
Cuando inicializar y cerrar la conexión desde un servlet . . . . .	33
Conexión dedicada . . . . .	34
Conectando a la Base de Datos . . . . .	34
CRUD básico . . . . .	35
LEER uno (findOne) . . . . .	35
LEER todos (findAll) . . . . .	36
Crear . . . . .	36
Actualizar . . . . .	37
Borrar . . . . .	38

CRUD (patrón DAO) . . . . .	38
InstalacionDao.java . . . . .	38
Leer todos/ uno (InstalacionDaoImpl.java) . . . . .	39
Crear (InstalacionDaoImpl.java) . . . . .	40
Actualizar (InstalacionDaoImpl.java) . . . . .	40
Borrar (InstalacionDaoImpl.java) . . . . .	42
<b>Añadiendo seguridad a la aplicación</b>	<b>42</b>
El servicio de gestión de sesión . . . . .	43
Creando la sesión . . . . .	43
Identificando sesión . . . . .	45
Cerrando sesión . . . . .	46
<b>WS: Servicios REST</b>	<b>47</b>
Single Application Page . . . . .	47
Creación de un WebService con Tomcat y Jersey . . . . .	48
Añadiendo las dependencias al proyecto . . . . .	48
Creando el POJO . . . . .	49
Patrón DAO . . . . .	50
El servicio web (WS) . . . . .	55
Probando el servicio . . . . .	59
<b>Aplicación híbrida</b>	<b>59</b>
Modificando el backend para que soporte peticiones de otro origen . . . . .	60
Preparación del entorno . . . . .	61
Ejemplo de aplicación HTML5/Javascript . . . . .	62
Localtunel . . . . .	65
Ejemplo 1: No más de una reserva al día por persona . . . . .	66
Ejemplo 2: No podemos reservar con más de dos semanas de antelación	67
<b>Comandos útiles de docker</b>	<b>69</b>

## Tema 02: Conectividad con Bases de Datos Relacionales

Con esta unidad integrada que vamos a trabajar en clase pretendemos cubrir los siguientes objetivos:

1. El desfase objeto-relacional.
2. Gestores de bases de datos embebidos e independientes.
3. Protocolos de acceso a bases de datos. Conectores.
4. Establecimiento de conexiones.
5. Definición de objetos destinados al almacenamiento del resultado de operaciones con bases de datos. Eliminación de objetos finalizada su función.
6. Ejecución de sentencias de descripción de datos.

7. Ejecución de sentencias de modificación de datos.
8. Ejecución de consultas.
9. Utilización del resultado de una consulta.
10. Ejecución de procedimientos almacenados en la base de datos.
11. Gestión de transacciones.

## Índice de contenido

1. Creación de la BBDD y CRUD básico.
2. Creación del proyecto en modo interactivo (MAVEN).
3. Introducción a los Servlets.
4. Patrones y estrategias de conexión.
5. Introducción a los Servlets.
6. Acceso identificado y gestión de sesiones.
7. Web Services con Jersey.
8. Procedimientos almacenados y disparadores.
9. APP híbrida para acceder al webservice.
10. Ayuda docker.

## El problema

Para aprender las dificultades a las que nos enfrentamos cuando tenemos que desarrollar una aplicación con un lenguaje orientado a objetos que almacene información en un sistema relacional, vamos a diseñar un sistema de gestión de reservas con usuarios e instalaciones configurables.

Vamos a ver el mismo problema desde dos enfoques diferentes:

1. **Enfoque clásico:** Con servlets y páginas JSP y gestión de sesiones y privilegios de acceso mediante filtros y cookies.
2. **Servicio REST:** Los end-points del backend son varios servicios implementados con Jersey con autenticación básica mediante filtros.

Con disparadores controlaremos temas como:

1. Que un usuario no pueda hacer más de una reserva en un día dado.
2. Que no se puedan hacer reservas con más de una semana de antelación.

Con filtros controlaremos que:

1. Sólo los usuarios identificados correctamente pueden hacer reservas
2. Un usuario no pueda modificar reservas de otro usuario
3. Sólo los usuarios identificados puedan hacer reservas
4. Los usuarios administradores son los únicos que pueden crear/actualizar/borrar usuarios

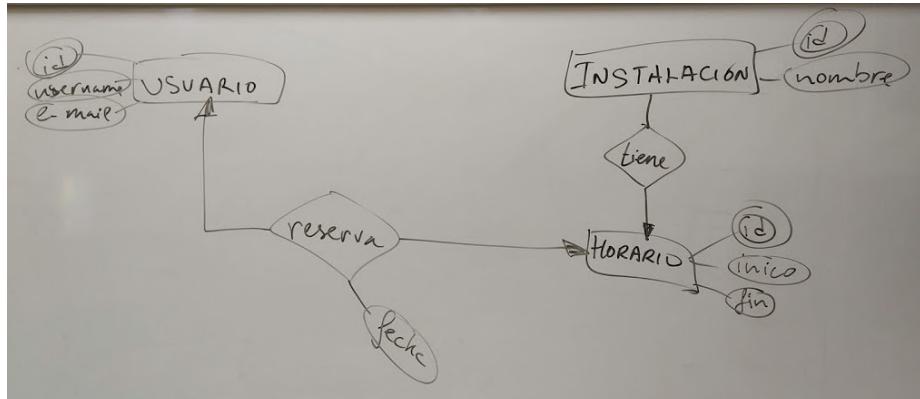


Figure 1: Diagrama ER

## Preparando el entorno

Para completar esta práctica necesitamos tener instalada una JDK (al menos 1.8 o superior), Tomcat 7 o superior (lo vamos a usar como dependencia Maven), MySQL Server (usaremos un contenedor Docker), Maven, Git, y como IDE: Microsoft Visual Studio Code (plugins Java Extension Pack, Java Code Generators, Tomcat for Java y MySQL -extensión por Jun Han-).

No es necesario, pero si usamos MySQL como contenedor Docker, es más cómodo usar Linux (si no, tendremos que cambiar la IP del host o servidor de la base de datos a la máquina virtual Linux en la que creamos los contenedores).

## Clonando el proyecto

```
1 $ git clone https://gitlab.iesvirgendelcarmen.com/juangu/tema02CRUD
```

## Creando los contenedores para la base de datos

Para crear los contenedores con la base de datos y una pequeña interfaz gráfica para poder ejecutar comandos SQL y examinar tablas y datos de manera sencilla, usamos el fichero `docs/stack.yml` de la siguiente manera:

```
1 $ docker-compose -f stack.yml up -d
```

Ten cuidado al manipular el fichero YML pues es un lenguaje de marcas tabulado, es decir, los bloques se anidan con tabulaciones.

Fíjate que en la opción “restart”. En producción deberás poner “always”, pero ahora para desarrollo lo dejamos en “no” para forzar nosotros cuando correr

o parar los contenedores con las órdenes (iniciar, parar, desactivar inicio automático con el anfitrión, activar inicio automático con el anfitrión):

```
1 $ docker start docker_adminer_1 docker_db_1
2 $ docker stop docker_adminer_1 docker_db_1
3 $ docker container update --restart=no docker_adminer_1 docker_db_1
4 $ docker container update --restart=yes docker_adminer_1 docker_db_1
```

Para ejecutar el proyecto simplemente desde el raíz del mismo hacemos:

```
1 $ mvn compile package
```

Para ejecutar tomcat directamente desde Maven:

```
1 mvn tomcat7:run
```

## Preparando el entorno

Para este proyecto puedes instalar MySQL directamente, usar un paquete como XAMPP en Windows/Linux/MAC, o bien con docker. La ventaja de usar Docker es que podemos eliminar todas las configuraciones y recrear todos los datos desde un comando

Aunque lo tienes en el directorio *docs*, para crear las aplicaciones docker usamos la siguiente receta:

```
1 # Use root/example as user/password credentials
2 version: '3.1'
3
4 services:
5
6 db:
7   image: mysql
8   command: --default-authentication-plugin=mysql_native_password
9   restart: "no"
10  volumes:
11    - ./bbdd.sql:/docker-entrypoint-initdb.d/bbdd.sql
12  environment:
13    MYSQL_ROOT_PASSWORD: example
14  ports:
15    - 33306:3306
16
17 adminer:
18   image: adminer
19   restart: "no"
20   ports:
21     - 8181:8080
```

## Creamos la BBDD

Fichero `bbdd.sql` para la creación de la base de datos y la carga inicial de datos (fíjate en el fichero `stack.yml`, cómo se carga este archivo *automágicamente* al crear las aplicaciones docker):

```
1 -- COMO ROOT PARA CREAR LA BBDD Y EL USUARIO
2
3 -- CREAMOS LA BBDD PARA UTF-8 COLLATION EN ESPAÑOL
4 CREATE DATABASE gestion_reservas CHARACTER SET utf16 COLLATE
   utf16_spanish_ci;
5
6 -- CAMBIAMOS LA BBDD ACTIVA
7 USE gestion_reservas;
8
9
10 -- CREAMOS LAS TABLAS
11 CREATE TABLE `usuario` (
12   `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
13   `username` varchar(12) NOT NULL,
14   `password` varchar(20) NOT NULL,
15   `email` varchar(50) NOT NULL
16 ) ENGINE='InnoDB';
17
18 CREATE TABLE `instalacion` (
19   `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
20   `nombre` varchar(50) NOT NULL
21 ) ENGINE='InnoDB';
22
23 CREATE TABLE `horario` (
24   `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
25   `instalacion` int(11) NOT NULL,
26   `inicio` time NOT NULL,
27   `fin` time NOT NULL,
28   FOREIGN KEY (`instalacion`) REFERENCES `instalacion` (`id`) ON
      DELETE CASCADE
29 ) ENGINE='InnoDB';
30
31 CREATE TABLE `reserva` (
32   `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
33   `usuario` int(11) NOT NULL,
34   `horario` int(11) NOT NULL,
35   `fecha` date NOT NULL,
36   FOREIGN KEY (`usuario`) REFERENCES `usuario` (`id`),
37   FOREIGN KEY (`horario`) REFERENCES `horario` (`id`)
38 ) ENGINE='InnoDB';
```

```

39
40 INSERT INTO `usuario` (`id`, `username`, `password`, `email`) VALUES
41 (2, 'pepillo', 'secreto', 'pepe@gmail.com'),
42 (5, 'admin', 'admin', 'admin@correo.com'),
43 (7, 'obijuan', 'starwars', 'darkside@starwars.com'),
44 (13, 'gerente', 'password1234', 'gerencia@vdc.com');
45
46 INSERT INTO `instalacion` (`id`, `nombre`) VALUES
47 (7, 'tenis arriba'),
48 (8, 'tenis césped artificial'),
49 (9, 'fútbol'),
50 (10, 'baloncesto'),
51 (11, 'squash'),
52 (13, 'sauna mujeres'),
53 (14, 'pista de pádel'),
54 (16, 'sauna caballeros');
55
56
57 INSERT INTO `horario` (`id`, `instalacion`, `inicio`, `fin`) VALUES
58 (1, 7, '08:00:00', '09:00:00'),
59 (2, 7, '09:00:00', '10:00:00'),
60 (3, 7, '10:00:00', '11:00:00'),
61 (4, 7, '11:00:00', '12:00:00'),
62 (5, 7, '12:00:00', '13:00:00'),
63 (6, 7, '13:00:00', '14:00:00'),
64 (7, 7, '14:00:00', '15:00:00'),
65 (8, 7, '15:00:00', '16:00:00'),
66 (9, 7, '16:00:00', '17:00:00'),
67 (10, 7, '17:00:00', '18:00:00'),
68 (11, 7, '18:00:00', '19:00:00'),
69 (12, 7, '19:00:00', '20:00:00'),
70 (13, 7, '20:00:00', '21:00:00'),
71 (14, 7, '21:00:00', '22:00:00'),
72 (15, 7, '22:00:00', '23:00:00'),
73 (16, 8, '08:00:00', '09:00:00'),
74 (17, 8, '09:00:00', '10:00:00'),
75 (18, 8, '10:00:00', '11:00:00'),
76 (19, 8, '11:00:00', '12:00:00'),
77 (20, 8, '12:00:00', '13:00:00'),
78 (21, 8, '13:00:00', '14:00:00'),
79 (22, 8, '14:00:00', '15:00:00'),
80 (23, 8, '15:00:00', '16:00:00'),
81 (24, 8, '16:00:00', '17:00:00'),
82 (25, 8, '17:00:00', '18:00:00'),
83 (26, 8, '18:00:00', '19:00:00'),
84 (27, 8, '19:00:00', '20:00:00'),

```

85 (28,	8,	'20:00:00',	'21:00:00'),
86 (29,	8,	'21:00:00',	'22:00:00'),
87 (30,	8,	'22:00:00',	'23:00:00'),
88 (31,	9,	'08:00:00',	'09:30:00'),
89 (32,	9,	'09:30:00',	'11:00:00'),
90 (33,	9,	'11:00:00',	'12:30:00'),
91 (34,	9,	'12:30:00',	'14:00:00'),
92 (35,	9,	'14:00:00',	'15:30:00'),
93 (36,	9,	'15:30:00',	'17:00:00'),
94 (37,	9,	'17:00:00',	'18:30:00'),
95 (38,	9,	'18:30:00',	'20:00:00'),
96 (39,	9,	'20:00:00',	'21:30:00'),
97 (40,	9,	'21:30:00',	'23:00:00'),
98 (41,	9,	'23:00:00',	'00:30:00'),
99 (42,	10,	'08:00:00',	'09:00:00'),
100 (43,	10,	'09:00:00',	'10:00:00'),
101 (44,	10,	'10:00:00',	'11:00:00'),
102 (45,	10,	'11:00:00',	'12:00:00'),
103 (46,	10,	'12:00:00',	'13:00:00'),
104 (47,	10,	'13:00:00',	'14:00:00'),
105 (48,	10,	'14:00:00',	'15:00:00'),
106 (49,	10,	'15:00:00',	'16:00:00'),
107 (50,	10,	'16:00:00',	'17:00:00'),
108 (51,	10,	'17:00:00',	'18:00:00'),
109 (52,	10,	'18:00:00',	'19:00:00'),
110 (53,	10,	'19:00:00',	'20:00:00'),
111 (54,	10,	'20:00:00',	'21:00:00'),
112 (55,	10,	'21:00:00',	'22:00:00'),
113 (56,	10,	'22:00:00',	'23:00:00'),
114 (57,	11,	'08:00:00',	'08:45:00'),
115 (58,	11,	'08:45:00',	'09:30:00'),
116 (59,	11,	'09:30:00',	'10:15:00'),
117 (60,	11,	'10:15:00',	'11:00:00'),
118 (61,	11,	'11:00:00',	'11:45:00'),
119 (62,	11,	'11:45:00',	'12:30:00'),
120 (63,	11,	'12:30:00',	'13:15:00'),
121 (64,	11,	'13:15:00',	'14:00:00'),
122 (65,	11,	'14:00:00',	'14:45:00'),
123 (66,	11,	'14:45:00',	'15:30:00'),
124 (67,	11,	'15:30:00',	'16:15:00'),
125 (68,	11,	'16:15:00',	'17:00:00'),
126 (69,	11,	'17:00:00',	'17:45:00'),
127 (70,	11,	'17:45:00',	'18:30:00'),
128 (71,	11,	'18:30:00',	'19:15:00'),
129 (72,	11,	'19:15:00',	'20:00:00'),
130 (73,	11,	'20:00:00',	'20:45:00'),

```

131 (74,    11, '20:45:00', '21:30:00'),
132 (75,    11, '21:30:00', '22:15:00'),
133 (103,   13, '09:00:00', '09:30:00'),
134 (104,   13, '09:30:00', '10:00:00'),
135 (105,   13, '10:00:00', '10:30:00'),
136 (106,   13, '10:30:00', '11:00:00'),
137 (107,   13, '11:00:00', '11:30:00'),
138 (108,   13, '11:30:00', '12:00:00'),
139 (109,   13, '12:00:00', '12:30:00'),
140 (110,   13, '12:30:00', '13:00:00'),
141 (111,   13, '13:00:00', '13:30:00'),
142 (112,   13, '13:30:00', '14:00:00'),
143 (113,   13, '14:00:00', '14:30:00'),
144 (114,   13, '14:30:00', '15:00:00'),
145 (115,   13, '15:00:00', '15:30:00'),
146 (116,   13, '15:30:00', '16:00:00'),
147 (117,   13, '16:00:00', '16:30:00'),
148 (118,   13, '16:30:00', '17:00:00'),
149 (119,   13, '17:00:00', '17:30:00'),
150 (120,   13, '17:30:00', '18:00:00'),
151 (121,   13, '18:00:00', '18:30:00'),
152 (122,   13, '18:30:00', '19:00:00'),
153 (123,   13, '19:00:00', '19:30:00'),
154 (124,   13, '19:30:00', '20:00:00'),
155 (125,   13, '20:00:00', '20:30:00'),
156 (126,   13, '20:30:00', '21:00:00'),
157 (127,   13, '21:00:00', '21:30:00'),
158 (128,   13, '21:30:00', '22:00:00'),
159 (129,   13, '22:00:00', '22:30:00'),
160 (130,   14, '08:00:00', '09:30:00'),
161 (131,   14, '09:30:00', '11:00:00'),
162 (132,   14, '11:00:00', '12:30:00'),
163 (133,   14, '12:30:00', '14:00:00'),
164 (134,   14, '14:00:00', '15:30:00'),
165 (135,   14, '15:30:00', '17:00:00'),
166 (136,   14, '17:00:00', '18:30:00'),
167 (137,   14, '18:30:00', '20:00:00'),
168 (138,   14, '20:00:00', '21:30:00'),
169 (139,   14, '21:30:00', '23:00:00'),
170 (140,   14, '23:00:00', '00:30:00);

171
172
173 INSERT INTO `reserva` (`id`, `usuario`, `horario`, `fecha`) VALUES
174 (1, 2, 130,      '2019-10-12'),
175 (2, 2, 130,      '2019-10-13'),
176 (4, 7, 120,      '2019-11-11'),

```

```
177 (5, 7, 130, '2019-11-21');
```

## Creación del proyecto en modo interactivo (MAVEN)

Si has clonado este repositorio, no es necesario hacer este paso, sólamente cuando quieras crear un proyecto como éste.

Para crear en modo interactivo el proyecto, estructura de directorios, fichero pom.xml, etc. **desde cero**, tendríamos que usar el proyecto Maven Java Web desde el IDE o bien desde línea de comandos ejecutaríamos esta instrucción:

```
1 $ mvn archetype:generate  
    -DarchetypeGroupId=org.apache.maven.archetypes \  
2      -DarchetypeArtifactId=maven-archetype-webapp  
        -DarchetypeVersion=1.4
```

Tras indicar el grupo (com.iesvdc.acceso.simplecrud) y el artefacto (simplecrud) Maven crea los ficheros necesarios.

## Dependencias Maven

Antes de comenzar, veamos las dependencias (librerías) adicionales que va a necesitar nuestro proyecto.

### MySQL

Necesitamos importar en la CLASS\_PATH del proyecto el driver JDBC de MySQL.

```
1 <dependency>  
2   <groupId>mysql</groupId>  
3   <artifactId>mysql-connector-java</artifactId>  
4   <version>8.0.1</version>  
5 </dependency>
```

### Soporte J2EE (Servlets)

Para poder usar JSP (Java Server Pages) y Servlets (las clases necesarias), tenemos que cargar la API Web de Java:

```
1 <dependency>
2   <groupId>javax</groupId>
3   <artifactId>javaee-web-api</artifactId>
4   <version>8.0.1</version>
5   <scope>provided</scope>
6 </dependency>
```

### Soporte para JSLT (para JSP)

Para poder usar las extensiones JSLT dentro de una página JSP, necesitamos importar la API JSLT:

```
1 <dependency>
2   <groupId>javax.servlet</groupId>
3   <artifactId>jstl</artifactId>
4   <version>1.2</version>
5 </dependency>
```

### Soporte JSON

Para hacer el marshalling/unmarshalling de objetos es muy sencillo usar la API Gson de Google. Ya vimos JAXB y Moxy, ahora exploramos otra opción.

```
1 <dependency>
2   <groupId>com.google.code.gson</groupId>
3   <artifactId>gson</artifactId>
4   <version>2.8.6</version>
5 </dependency>
```

### Insertando el servidor Tomcat en Maven

Para poder ejecutar Tomcat desde maven para no necesitar descargar e instalar el servidor de aplicaciones Java de la Apache foundation, añadimos estas líneas dentro de “build->plugins”:

```
1 <plugin>
2   <groupId>org.apache.tomcat.maven</groupId>
3   <artifactId>tomcat7-maven-plugin</artifactId>
4   <version>2.2</version>
5   <configuration>
6     <port>9090</port>
7     <path>/</path>
8   </configuration>
9 </plugin>
```

## Servlets

Un servlet es un programa Java que se ejecuta en un servidor (normalmente de HTTP) y extiende su funcionalidad. Atiende peticiones recibidas desde los clientes y genera las respuestas.

### Ciclo de vida

#### Inicializar el servlet

Cuando un servidor carga un servlet, ejecuta el **método init del servlet**. El proceso de inicialización debe completarse antes de poder manejar peticiones de los clientes, y antes de que el servlet sea destruido.

Aunque muchos servlets se ejecutan en servidores multi-thread, los servlets no tienen problemas de concurrencia durante su inicialización. El servidor llama sólo una vez al método init al crear la instancia del servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet. El servidor no puede recargar un servlet sin primero haber destruido el servlet llamando al método destroy.

#### Interactuar con los clientes

Después de la inicialización, el servlet puede dar servicio a las peticiones de los clientes. Estas peticiones serán atendidas por la misma instancia del servlet, por lo que hay que tener cuidado al acceder a variables compartidas, ya que podrían darse problemas de sincronización entre requerimientos simultáneos. Hablamos, entre otros, de los métodos:

1. doGet
2. doPost
3. doPut
4. doDelete

#### Destruir el servlet

Los servlets se ejecutan hasta que el servidor los destruye, por cierre del servidor o bien a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método destroy del propio servlet. Este método sólo se ejecuta una vez y puede ser llamado cuando aún queden respuestas en proceso, por lo que hay que tener la atención de esperarlas. El servidor no ejecutará de nuevo el servlet hasta haberlo cargado e inicializado de nuevo.

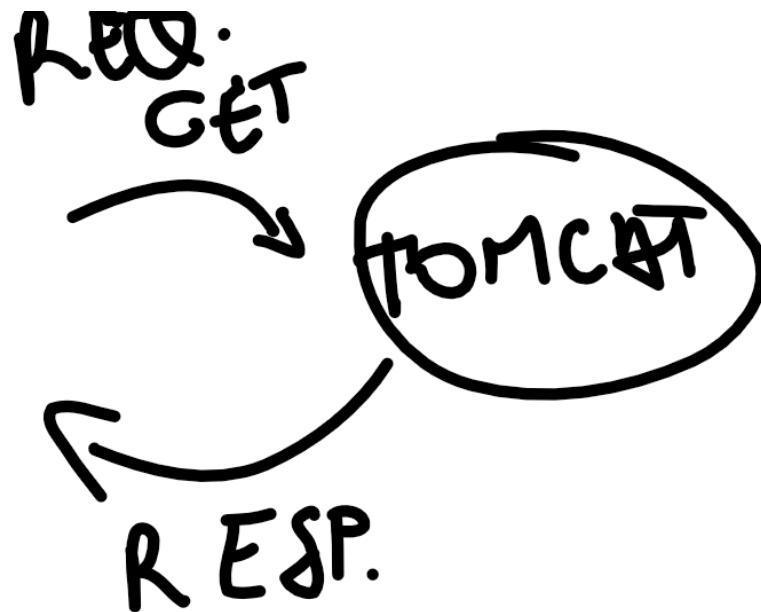
En el código de la clase es el **método destroy**.

## ¿Cómo funciona un servlet?

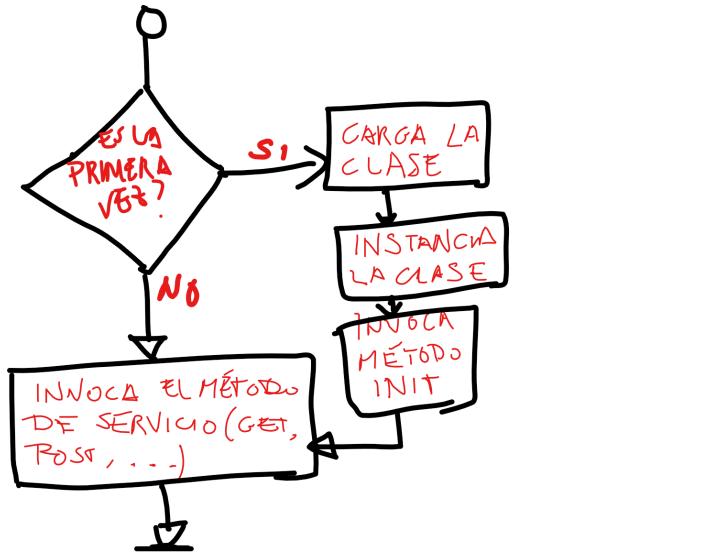
Hasta ahora hemos creado aplicaciones J2SE, es decir aplicaciones para ejecutar en el equipo local.

Un **servlet** por sí mismo no tiene capacidad de ejecutarse, no tenemos un método *main* como hasta ahora.

El **servlet** se ejecuta dentro de un **contenedor** que es un servidor de aplicaciones. El contenedor será la verdadera aplicación que responde a las peticiones de los clientes (por ejemplo servicios REST, HTTP, etc.).



En función de *la ruta de acceso al recurso* el servidor de aplicaciones cargará una u otra aplicación.



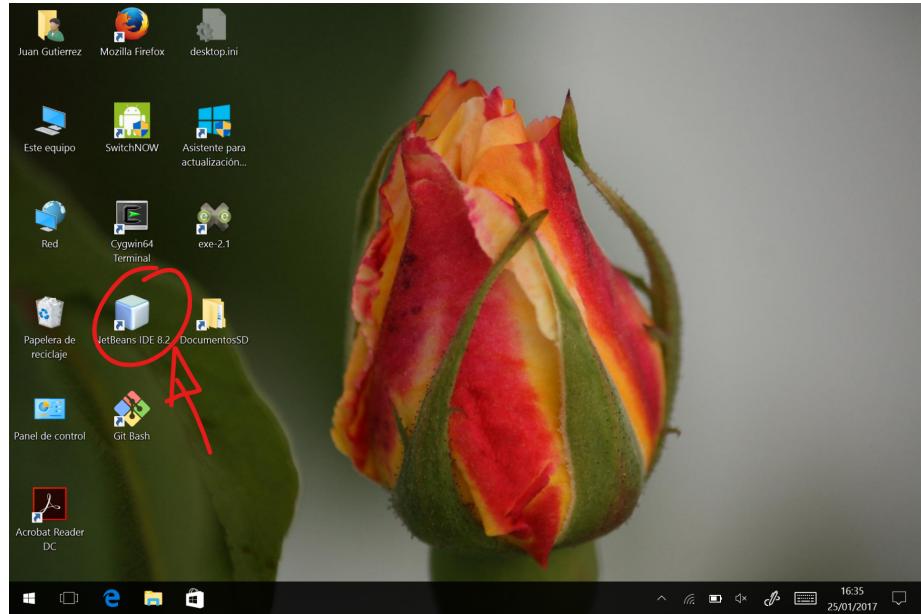
La primera vez que se invoque la ruta de nuestro servlet, éste será cargado e inicializado. Las siguientes veces ya estará en memoria y un hilo de ejecución invocará el método correspondiente.

Method Name	Description
doGet	Used to process HTTP GET requests. Input sent to the servlet must be included in the URL address. For example: ?myName=Josh&myBook=JavaEERecipes.
doPost	Used to process HTTP POST requests. Input can be sent to the servlet within HTML form fields.
doPut	Used to process HTTP PUT requests.
doDelete	Used to process HTTP DELETE requests.
doHead	Used to process HTTP HEAD requests.
doOptions	Called by the container to allow OPTIONS request handling.
doTrace	Called by the container to handle TRACE requests.
getLastModified	Returns the time that the <code>HttpServletRequest</code> object was last modified.
init	Initializes the servlet.
destroy	Finalizes the servlet.
getServletInfo	Provides information regarding the servlet.

## Preparando el entorno

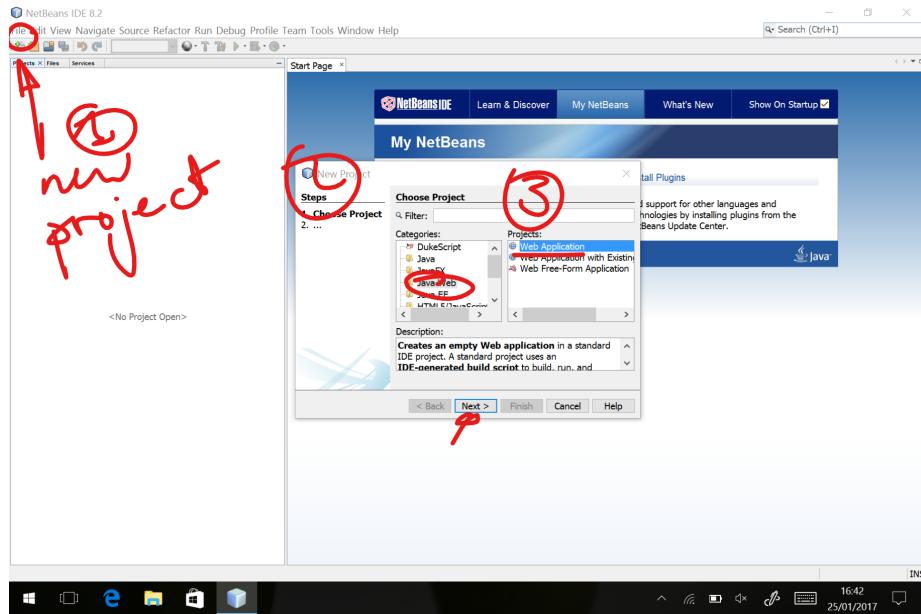
Para crear el proyecto vamos a usar Netbeans IDE y el servidor de aplicaciones J2EE Tomcat.

Partimos que ya hemos descargado e instalado NetBeans.

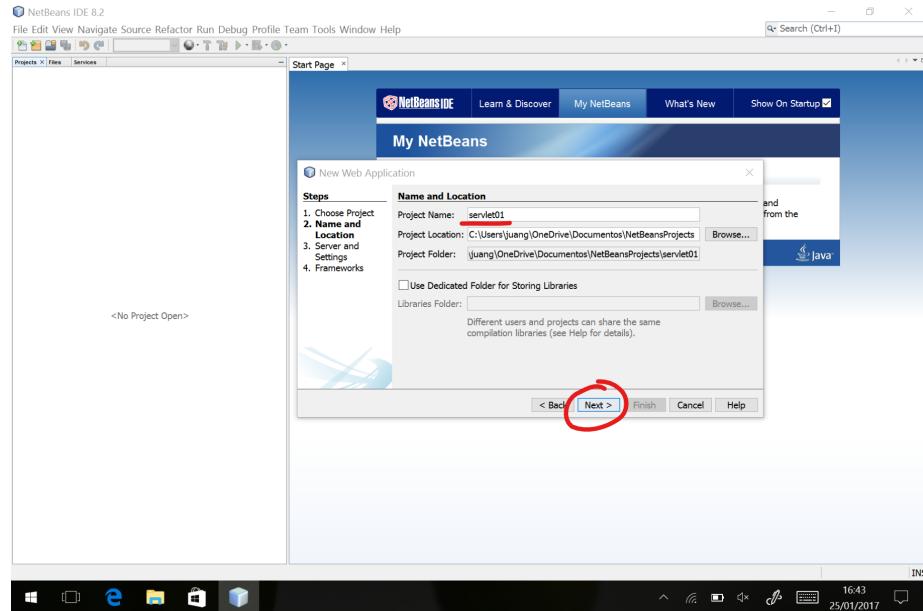


## Creando el proyecto

Seleccionamos el menú File-> New project.



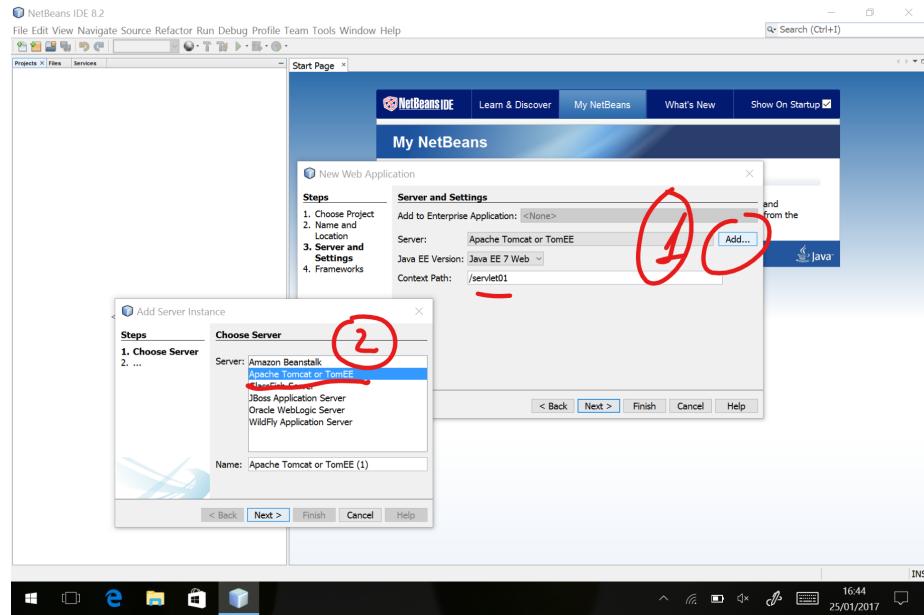
Ahora le damos un nombre al proyecto y seleccionamos dónde queremos guardarlo en nuestro disco duro. Pulsamos **Next**.



### Selección del contenedor(servidor) J2EE.

Como es una aplicación J2EE, necesitamos instalar un servidor de aplicaciones. Por defecto NetBeans ofrece GlassFish, pero nosotros vamos a ver aquí:

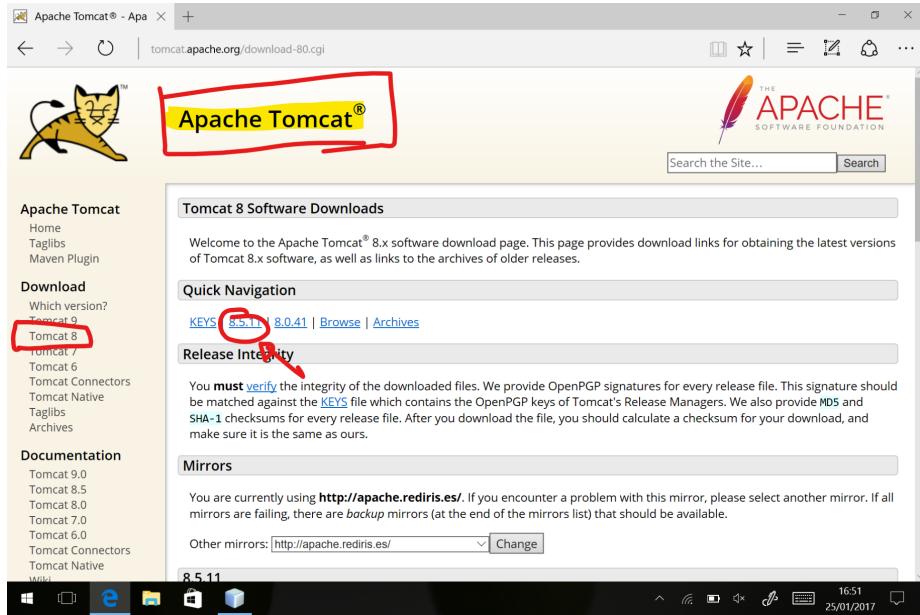
Apache Tomcat



### Descarga e instalación de Tomcat.

Antes de continuar, deberemos descargar y configurar Apache Tomcat de su Web oficial.

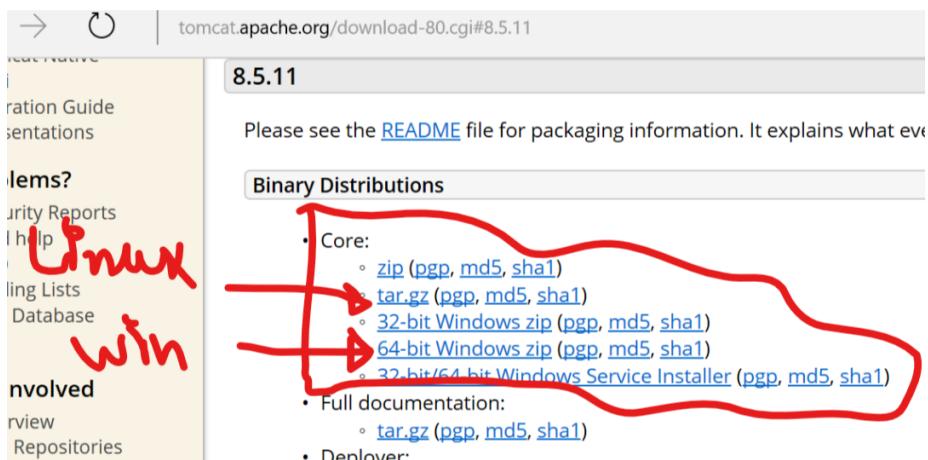
En nuestro caso nos decantamos por la versión 8.5.XX.



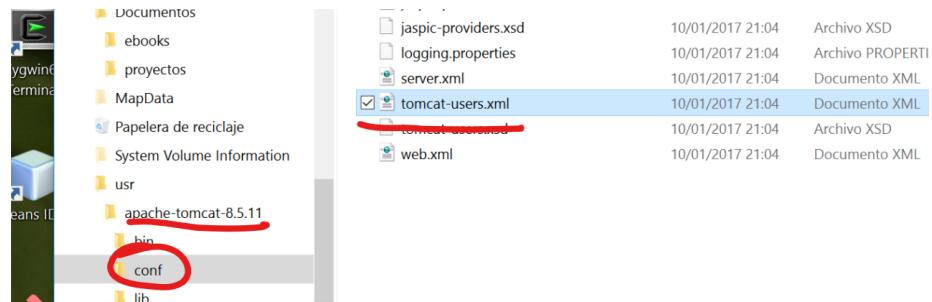
Seleccionamos el “tar.gz” si estamos en un sistema \*NIX o MAC y el ZIP en Windows.

Descargamos y descomprimimos en nuestra carpeta de trabajo. No es necesaria instalación y es mejor así para mantener el equipo más limpio.

Si quieras desacerte de Tomcat al terminar el tutorial, sólo has de eliminar la carpeta y ¡listo!.



Una vez descomprimido, hemos de configurar el acceso de administrador para poder subir aplicaciones (ficheros WAR) al servidor. Para ello editamos el fichero **tomcat-users.xml** que está en la carpeta **conf**.



Añadimos la siguiente línea en el fichero **tomcat-users.xml**. Asegúrate de hacerlo dentro del nodo raíz “tomcat-users” del fichero XML:

```
1 <user username="tomcat" password="tomcat" roles="admin,
  manager-gui"/>
```

The screenshot shows the NetBeans code editor with the file 'tomcat-users.xml' open. The code is as follows:

```

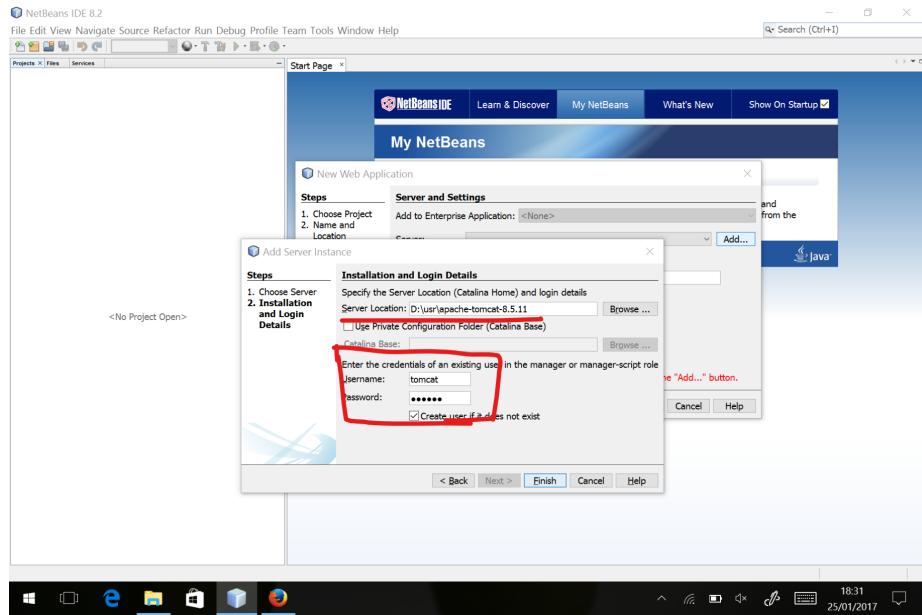
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3      Licensed to the Apache Software Foundation (ASF) under one or more
4      contributor license agreements. See the NOTICE file distributed with
5      this work for additional information regarding copyright ownership.
6      The ASF licenses this file to You under the Apache License, Version 2.0
7      (the "License"); you may not use this file except in compliance with
8      the License. You may obtain a copy of the License at
9
10     http://www.apache.org/licenses/LICENSE-2.0
11
12     Unless required by applicable law or agreed to in writing, software
13     distributed under the License is distributed on an "AS IS" BASIS,
14     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15     See the License for the specific language governing permissions and
16     limitations under the License.
17   -->
18   <tomcat-users xmlns="http://tomcat.apache.org/xml"
19                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
20                 xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xs
21                 version="1.0">
22
23     <user username="tomcat" password="tomcat" roles="manager-gui,admin"/>
24   <!--
25       NOTE: By default, no user is included in the "manager-gui" role required
26       to operate the "/manager/html" web application. If you wish to use this app

```

A red box highlights the line containing the user definition: <user username="tomcat" password="tomcat" roles="manager-gui,admin"/>.

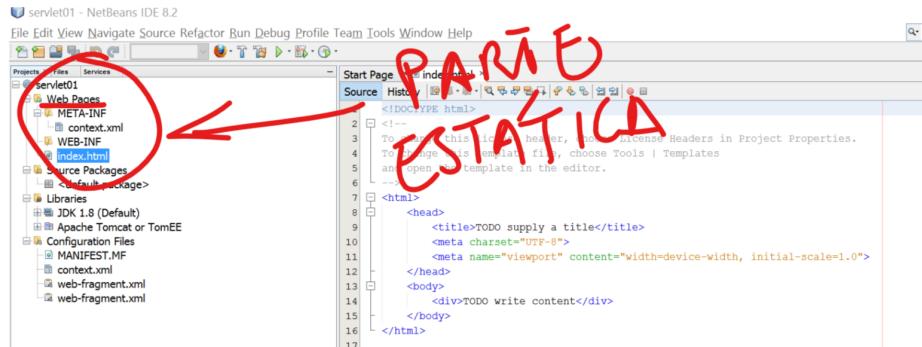
### Introducimos en NetBeans el usuario y contraseña de Tomcat

Continuamos con el proyecto: indicamos a NetBeans el directorio donde hemos descargado y configurado Tomcat así como el usuario y contraseña para el acceso al servidor de aplicaciones.



## Estructura de nuestro proyecto Web con Servlets

En el directorio **Web Pages** podemos colgar cualquier elemento estático que necesitemos (código JavaScript, CSS, formularios estáticos, etc.).



## Añadiendo el primer servlet

El fichero web.xml:

```

1  <!-- Página por defecto si no ponemos nada -->
2  <welcome-file-list>
3      <welcome-file>index.jsp</welcome-file>
4  </welcome-file-list>
```

```

5      <!-- Servlet para la gestión de usuarios -->
6      <servlet>
7          <!-- Es como una variable, le asigno un nombre
8              a una clase que es la que contiene mi servlet -->
9          <servlet-name>HorarioManagement</servlet-name>
10         <!-- Aquí le damos la ruta completa (classpath),
11             incluyendo el paquete -->
12         <servlet-class>com.iesvdc.acceso.simplecrud.controller.HorarioManagement</servlet-class>
13     </servlet>
14     <!-- Mapeo Ruta <==> Servlet -->
15     <servlet-mapping>
16         <servlet-name>HorarioManagement</servlet-name>
17         <!-- Ruta en el servicio Web -->
18         <url-pattern>/user</url-pattern>
19         <url-pattern>/user/*</url-pattern>
20     </servlet-mapping>
21 
```

Ejemplo de un servlet:

```

1 package com.iesvdc.acceso.simplecrud.controller;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10 public class horarioManagement extends HttpServlet {
11
12     @Override
13     public void init() throws ServletException {
14
15     }
16
17     // LEER
18     @Override
19     protected void doGet(HttpServletRequest req, // parámetros de la
20                         // petición
21                         HttpServletResponse resp) // respuesta que genero
22                         throws ServletException, IOException {
23
24     }
25
26     // CREAR
27 
```

```

26     @Override
27     protected void doPost(HttpServletRequest req, // parámetros de
28                           la petición
29                           HttpServletResponse resp) // respuesta que genero
30                           throws ServletException, IOException {
31 }
32
33 // ACTUALIZAR
34 @Override
35     protected void doPut(HttpServletRequest req, // parámetros de la
36                           petición
37                           HttpServletResponse resp) // respuesta que genero
38                           throws ServletException, IOException {
39 }
40
41
42 // BORRAR
43 @Override
44     protected void doDelete(HttpServletRequest req, // parámetros de
45                           la petición
46                           HttpServletResponse resp) // respuesta que genero
47                           throws ServletException, IOException {
48 }
49
50 @Override
51     public void destroy() {
52 }
53
54 }
55 }
```

## Conexión a la base de datos

### Creando los Plain Old Java Objects

Para poder hacer el marshalling/unmarshalling de objetos, te recomendamos primero tener los objetos en Java, es lo que llamamos el modelo de datos, en algunos sitios llamados clases entidad y en Spring Java POJOs (Plain Old Java Object).

A tal efecto creamos clases sencillas como Usuario, Instalacion, Reserva, etc.

con sus correspondientes constructores, getters y setters.

Usuario.java (ten cuidado con los métodos *equals* para comparar dos usuarios):

```
1 package com.iesvdc.acceso.simplecrud.model;
2
3 public class Usuario {
4
5     String username;
6     String email;
7     String password;
8     Integer id;
9
10    Usuario() {
11
12    }
13
14    Usuario(String username, String email, String password){
15        this.username = username;
16        this.password = password;
17        this.email = email;
18    }
19
20    Usuario(Integer id, String username, String email, String
21            password){
22        this.id = id;
23        this.username = username;
24        this.password = password;
25        this.email = email;
26    }
27
28    public String getUsername() {
29        return this.username;
30    }
31
32    public void setUsername(String username) {
33        this.username = username;
34    }
35
36    public String getEmail() {
37        return this.email;
38    }
39
40    public void setEmail(String email) {
41        this.email = email;
42    }
```

```

43
44     public String getPassword() {
45         return this.password;
46     }
47
48     public void setPassword(String password) {
49         this.password = password;
50     }
51
52     public Integer getId() {
53         return this.id;
54     }
55
56     public void setId(Integer id) {
57         this.id = id;
58     }
59
60     @Override
61     public String toString() {
62         return "{" +
63             " username='" + getUsername() + "' " +
64             ", email='" + getEmail() + "' " +
65             ", password='" + getPassword() + "' " +
66             ", id='" + getId() + "' " +
67             "}";
68     }
69
70     @Override
71     public boolean equals(Object o) {
72         if (o == this)
73             return true;
74         if (!(o instanceof Usuario)) {
75             return false;
76         }
77
78         Usuario u = (Usuario) o;
79
80         return u.getId() == this.id &&
81             u.getUsername().compareTo(this.username)==0 &&
82             u.getPassword().compareTo(this.password)==0 &&
83             u.getEmail().compareTo(this.email)==0;
84     }
85 }
```

Instalación (ten cuidado con los métodos *equals*):

```

1 package com.iesvdc.acceso.simplecrud.model;
2
3 public class Instalacion {
4
5     private int id;
6     private String name;
7
8     public Instalacion(){}
9
10    public Instalacion(String name) {
11        this.name = name;
12    }
13
14    public Instalacion(int id, String name) {
15        this.id = id;
16        this.name = name;
17    }
18
19    public int getId() {
20        return id;
21    }
22
23    public void setId(int id) {
24        this.id = id;
25    }
26
27    public String getName() {
28        return name;
29    }
30
31    public void setName(String name) {
32        this.name = name;
33    }
34
35    @Override
36    public boolean equals(Object o) {
37        if (o == this)
38            return true;
39        if (!(o instanceof Instalacion)) {
40            return false;
41        }
42        Instalacion instalacion = (Instalacion) o;
43
44        return instalacion.id == this.id &&
45               instalacion.name.compareTo(this.name)==0;
46    }

```

```

47
48     @Override
49     public String toString() {
50         return "{" +
51             " id='" + getId() + "'" +
52             ", name='" + getName() + "'" +
53             "}";
54     }
55
56
57 }
```

Horario (ten cuidado con los métodos *equals* para comparaciones):

```

1 package com.iesvdc.acceso.simplecrud.model;
2
3 import java.time.LocalTime;
4
5 public class Horario {
6     Instalacion instalacion;
7     LocalTime inicio;
8     LocalTime fin;
9     Long id;
10
11    public Horario() {
12    }
13
14    public Horario(Long id, Instalacion instalacion, LocalTime
15                  inicio, LocalTime fin) {
16        this.instalacion = instalacion;
17        this.inicio = inicio;
18        this.fin = fin;
19        this.id = id;
20    }
21
22    public Instalacion getInstalacion() {
23        return this.instalacion;
24    }
25
26    public void setInstalacion(Instalacion instalacion) {
27        this.instalacion = instalacion;
28    }
29
30    public LocalTime getInicio() {
31        return this.inicio;
32    }
33}
```

```

32
33     public void setInicio(LocalTime inicio) {
34         this.inicio = inicio;
35     }
36
37     public LocalTime getFin() {
38         return this.fin;
39     }
40
41     public void setFin(LocalTime fin) {
42         this.fin = fin;
43     }
44
45     public Long getId() {
46         return this.id;
47     }
48
49     public void setId(Long id) {
50         this.id = id;
51     }
52
53     public Horario instalacion(Instalacion instalacion) {
54         this.instalacion = instalacion;
55         return this;
56     }
57
58     public Horario inicio(LocalTime inicio) {
59         this.inicio = inicio;
60         return this;
61     }
62
63     public Horario fin(LocalTime fin) {
64         this.fin = fin;
65         return this;
66     }
67
68     public Horario id(Long id) {
69         this.id = id;
70         return this;
71     }
72
73     @Override
74     public boolean equals(Object o) {
75         if (o == this)
76             return true;
77         if (!(o instanceof Horario)) {

```

```

78         return false;
79     }
80     Horario horario = (Horario) o;
81
82     return horario.id == this.id &&
83         horario.inicio.compareTo(this.inicio)==0 &&
84         horario.fin.compareTo(this.fin)==0 &&
85         horario.instalacion.equals(this.instalacion);
86 }
87
88
89     @Override
90     public String toString() {
91         return "{" +
92             " instalacion='"+ getInstalacion().toString() + "'"+ +
93             ", inicio='"+ getInicio() + "'"+ +
94             ", fin='"+ getFin() + "'"+ +
95             ", id='"+ getId() + "'"+ +
96             "}";
97     }
98
99
100 }
```

Reserva (ten cuidado con los métodos *equals*):

```

1 package com.iesvdc.acceso.simplecrud.model;
2
3 import java.sql.Date;
4
5 public class Reserva {
6     Long id;
7     Usuario usuario;
8     Horario horario;
9     Date fecha;
10
11
12     public Reserva() {
13     }
14
15     public Reserva(Usuario usuario, Horario horario, Date fecha) {
16         this.usuario = usuario;
17         this.horario = horario;
18         this.fecha = fecha;
19     }
20 }
```

```
21  public Reserva(Long id, Usuario usuario, Horario horario, Date
22    fecha) {
23      this.id = id;
24      this.usuario = usuario;
25      this.horario = horario;
26      this.fecha = fecha;
27  }
28
29  public Long getId() {
30      return this.id;
31  }
32
33  public void setId(Long id) {
34      this.id = id;
35  }
36
37  public Usuario getUsuario() {
38      return this.usuario;
39  }
40
41  public void setUsuario(Usuario usuario) {
42      this.usuario = usuario;
43  }
44
45  public Horario getHorario() {
46      return this.horario;
47  }
48
49  public void setHorario(Horario horario) {
50      this.horario = horario;
51  }
52
53  public Date getFecha() {
54      return this.fecha;
55  }
56
57  public void setFecha(Date fecha) {
58      this.fecha = fecha;
59  }
60
61  public Reserva id(Long id) {
62      this.id = id;
63      return this;
64  }
65
66  public Reserva usuario(Usuario usuario) {
```

```

66     this.usuario = usuario;
67     return this;
68 }
69
70 public Reserva horario(Horario horario) {
71     this.horario = horario;
72     return this;
73 }
74
75 public Reserva fecha(Date fecha) {
76     this.fecha = fecha;
77     return this;
78 }
79
80 @Override
81 public boolean equals(Object o) {
82     if (o == this)
83         return true;
84     if (!(o instanceof Reserva))
85         return false;
86     Reserva reserva = (Reserva) o;
87     return usuario.equals(reserva.usuario) &&
88             horario.equals(reserva.horario) &&
89             fecha.compareTo(reserva.fecha)!=0;
90 }
91
92
93
94
95 @Override
96 public String toString() {
97     return "{" +
98         " id='" + getId() + "' " +
99         " usuario='" + this.getUsuario().toString() + "' " +
100        ", horario='" + this.getHorario().toString() + "' " +
101        ", fecha='" + getFecha().toString() + "' " +
102        "}";
103 }
104
105
106 }

```

Ahora ya sería posible con Gson o JAXB por ejemplo, desde el servlet directamente hacer el marshalling/unmarshalling de JSON a Java. No obstante no recomendamos esta opción porque no responde a ningún estándar como sí lo hace por ejemplo Jersey (JAX-RS 2.1, que es la especificación de la JSR 370:

JavaTM API for RESTful Web Services).

## Conectando a la base de datos cargando la configuración vía JNDI

En un entorno genérico, si por ejemplo queremos usar el patrón singleton y usar un único objeto conexión en nuestro código, podríamos hacer lo siguiente:

Abrir la conexión:

```
1 public class Conexion {
2     Connection conn;
3     public Conexion(){
4         if (conn==null)
5             try {
6                 Class.forName("com.mysql.jdbc.Driver");
7                 this.conn =
8                     DriverManager.getConnection("jdbc:mysql://localhost/gestion_reservas?" +
9                         "useUnicode=true&useJDBCCompliantTimezoneShift=true&serverTimezone=UTC" +
10                        "&user=root&password=example");
11             } catch (SQLException | ClassNotFoundException ex) {
12                 Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,
13                                         null, ex);
14             }
15         public Connection getConnection() {
16             return conn;
17         }
18 }
```

Para cargar la conexión por JNDI (Java Naming and Directory Interface), es decir pedimos a Java que localice, en el contexto actual, un objeto que será la información de la conexión a la base de datos, lo haríamos así:

```
1 import java.sql.Connection;
2 import java.sql.SQLException;
3 import javax.naming.Context;
4 import javax.naming.InitialContext;
5 import javax.sql.DataSource;
6
7 /**
8 * 
9 * @author juangu
10 */
11 public class Conexion {
```

```

13     Connection conn;
14     Context ctx;
15     DataSource ds;
16
17     public Conexion() {
18         // Vía DataSource con Contexto injectado
19         try {
20             if (ctx == null)
21                 ctx = new InitialContext();
22             if (ds == null)
23                 ds = (DataSource) ((Context) ctx.lookup(
24                     "java:comp/env")).lookup("jdbc/gestionReservas");
25             conn = ds.getConnection();
26         } catch (Exception ex) {
27             System.out.println("## Conexion ERROR ## " +
28                 ex.getLocalizedMessage());
29             ctx = null;
30             ds = null;
31             conn = null;
32         }
33     }
34
35     public Connection getConnection() {
36         return conn;
37     }

```

Esto implica que en el directorio `src/main/webapp/META-INF` de nuestro proyecto Maven, deberíamos tener un fichero llamado **context.xml** con el siguiente contenido:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Context path="/">
3     <Resource name="jdbc/gestionReservas"
4             global="jdbc/gestionReservas"
5             url="jdbc:mysql://localhost:33306/gestion_reservas"
6             auth="Container" type="javax.sql.DataSource"
7             maxTotal="100" maxIdle="30" maxWaitMillis="10000"
8             username="root" password="example"
9             driverClassName="com.mysql.cj.jdbc.Driver"
10        />
11 </Context>

```

En este archivo podemos ver que el recurso será accesible vía JDNI bajo el nombre “`jdbc/gestionReservas`”, que es como tener un objeto en memoria con las propiedades siguientes:

```

1  gestionReservas: {
2      url:"jdbc:mysql://localhost:33306/gestion_reservas",
3      auth:"Container",
4      type:"javax.sql.DataSource",
5      maxTotal:"100",
6      maxIdle:"30",
7      maxWaitMillis:"10000",
8      username:"root",
9      password:"example",
10     driverClassName:"com.mysql.cj.jdbc.Driver"
11 }

```

## Cuando inicializar y cerrar la conexión desde un servlet

Cuando hemos de tomar la decisión de cuando conectar a la base de datos desde un servlet, nos encontramos frente cuatro opciones:

1. **Conexión por transacción:** dentro de cada método doGet, doPost, doPut o doDelete abrimos y cerramos la conexión. En el método init() del servlet cargamos el driver JDBC correspondiente (Oracle, MySQL, PostgreSQL, etc.). Éste es el modelo que seguiremos en los servicios REST, no es el más óptimo pero sí es seguro. Además cuando saltemos a JPA con Hibernate en Spring, nos resultará más fácil el cambio.
2. **Conexión dedicada:** al crear el servlet abrimos la conexión (método init() del mismo) y se cierra al descargar el servlet (método destroy()). Por tanto el driver y la conexión se cargan en el método init().
3. **Conexión por sesión:** cargamos el driver JDBC necesario en el método init(). No abrimos la conexión hasta el primer do(Get|Put|Delete|Post). En la sesión de usuario vamos pasando la conexión abierta de unos métodos a otros.
4. **Conexión cacheada:** Con un “pool” de conexiones. El servidor de aplicaciones (Tomcat, Jetty, Glassfish...) es el encargado de cargar el driver y abrir la conexión la primera vez que se necesita y ofrecerla a cada servlet que la necesita. Hay que crear el “pool” de conexiones en el servidor, lo que implica que tenemos acceso a su administración. Si compartimos el servidor o bien conectamos a diferentes bases de datos, puede ser peligroso tocar las configuraciones porque podemos dejar al resto de aplicaciones del servidor sin conexión a su base de datos.

Aunque lo normal es delegar en el servidor de aplicaciones la gestión de conexiones al servidor de base de datos, una receta muy común es abrir y cerrar la conexión desde los métodos init() y destroy() de los mismos. Esto es lo que se llama una **conexión dedicada**. Veámoslo en los siguientes ejemplos.

## Conexión dedicada

```
1 public class Alumno extends HttpServlet {  
2  
3     Connection conn;  
4  
5     @Override  
6     public void init() throws ServletException {  
7         Conexion conexion = new Conexion();  
8         this.conn = conexion.getConnection();  
9     }  
}
```

```
1 @Override  
2     public void destroy() {  
3         try {  
4             this.conn.close();  
5         } catch (SQLException ex) {  
6             //  
7         }  
8     }  
}
```

## Conectando a la Base de Datos

Abrir la conexión:

```
1 public class Conexion {  
2     Connection conn;  
3     public Conexion(){  
4         if (conn==null)  
5             try {  
6                 Class.forName("com.mysql.jdbc.Driver");  
7                 conn =  
8                     DriverManager.getConnection("jdbc:mysql://localhost/gestion_reservas?"  
9                         +  
10                        "useUnicode=true&useJDBCCompliantTimezoneShift=true&serverTimezone=UTC"+  
11                        "&user=root&password=example");  
12             } catch (SQLException | ClassNotFoundException ex) {  
13                 Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,  
14                                         null, ex);  
15             }  
16     }  
17 }
```

```

15     public Connection getConnection() {
16         return conn;
17     }
18 }
```

## CRUD básico

### LEER uno (findOne)

```

1   String jsonObject="{}";
2   Connection conexion;
3   PreparedStatement pstm;
4   String jdbcURL;
5
6   jdbcURL = JDBC_MYSQL_GESTION_RESERVAS;
7
8   try {
9       Class.forName("com.mysql.cj.jdbc.Driver");
10      conexion = DriverManager.getConnection(jdbcURL, "root",
11          "example");
12      String sql = "SELECT * FROM usuario WHERE id=?";
13      pstm = conexion.prepareStatement(sql);
14      pstm.setInt(1, Integer.parseInt(id));
15      ResultSet rs = pstm.executeQuery();
16      if ( rs.next() ) {
17          String username = rs.getString("username");
18          String password = rs.getString("password");
19          String email = rs.getString("email");
20          // String id = rs.getString("id");
21          jsonObject = "{" + "\n" +
22              "'id': '" + id + "', " + "\n" +
23              "'username': '" + username + "', " + "\n" +
24              "'password': '" + password + "', " + "\n" +
25              "'email': '" + email + "'"+ "\n" +
26              "}";
27      }
28  }catch(Exception ex){
29      // Gestión de la excepción
30  }
// devolvemos jsonObject
```

## LEER todos (findAll)

```
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
2 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8" %>
4
5 <sql:query var="userList" dataSource="jdbc/gestionReservas">
6   select username, email, password from usuario;
7 </sql:query>
8
9
10 <%@ include file="header.jsp" %>
11   <div align="center">
12     <table border="1" cellpadding="5">
13       <caption><h2>Lista de alumnos</h2></caption>
14       <tr>
15         <th>username</th>
16         <th>email</th>
17         <th>password</th>
18       </tr>
19       <c:forEach var="usuario" items="${userList.rows}">
20         <tr>
21           <td><c:out value="${usuario.username}" /></td>
22           <td><c:out value="${usuario.email}" /></td>
23           <td><c:out value="${usuario.password}" /></td>
24         </tr>
25       </c:forEach>
26     </table>
27   </div>
28 <%@ include file="footer.jsp"%>
```

## Crear

```
1 Connection conexion;
2 PreparedStatement pstm;
3 String jdbcURL;
4 jdbcURL = JDBC_MYSQL_GESTION_RESERVAS;
5 try {
6   Class.forName("com.mysql.cj.jdbc.Driver");
7   conexion = DriverManager.getConnection(jdbcURL, "root",
8   "example");
9   String sql = "INSERT INTO usuario (username,password,email)
   VALUES(?, ?, ?)";
```

```

9      pstm = conexion.prepareStatement(sql);
10     pstm.setString(1, username);
11     pstm.setString(2, password);
12     pstm.setString(3, email);
13     if (pstm.executeUpdate() >0) {
14         // "Usuario insertado"
15     } else {
16         // "No se ha podido insertar"
17     }
18     conexion.close();
19 } catch (Exception ex) {
20     // "Imposible conectar a la BBDD"
21 }
```

## Actualizar

```

1 Usuario user = new Gson().fromJson(req.getReader(),
2                                         Usuario.class);
3 String jdbcURL = JDBC_MYSQL_GESTION_RESERVAS;
4 try {
5     Class.forName("com.mysql.cj.jdbc.Driver");
6     Connection conexion = DriverManager.getConnection(jdbcURL,
7             "root", "example");
8     String sql = "UPDATE usuario SET username=?, password=?,
9                 email=? WHERE id=?";
10    PreparedStatement pstm = conexion.prepareStatement(sql);
11    pstm.setString(1, user.getUsername());
12    pstm.setString(2, user.getPassword());
13    pstm.setString(3, user.getEmail());
14    pstm.setInt(4, user.getId());
15
16    if (pstm.executeUpdate() >0) {
17        resp.getWriter().println("Usuario insertado");
18    } else {
19        resp.getWriter().println("No se ha podido insertar");
20    }
21
22    conexion.close();
23 } catch (Exception ex) {
24     resp.getWriter().println(ex.getMessage());
25     resp.getWriter().println(ex.getLocalizedMessage());
26     // resp.getWriter().println("Imposible conectar a la BBDD");
27 }
```

## Borrar

```
1 Connection conexion;
2 PreparedStatement pstm;
3 String jdbcURL;
4
5 jdbcURL = JDBC_MYSQL_GESTION_RESERVAS;
6 try {
7     Class.forName("com.mysql.cj.jdbc.Driver");
8     conexion = DriverManager.getConnection(jdbcURL, "root",
9         "example");
10    String sql = "DELETE FROM usuario WHERE id=?";
11    pstm = conexion.prepareStatement(sql);
12    pstm.setInt(1, Integer.parseInt(id));
13    if ( pstm.executeUpdate()==0 ) {
14        jsonObject="{ "+
15            "'id':'"+id+"'}";
16    }
17 }catch(Exception ex){
18     // "No se pudo eliminar"
19 }
```

## CRUD (patrón DAO)

Ya tenemos las clases base que contiene los objetos que vamos a almacenar/re recuperar de la base de datos, ahora hay que hacer las interfaces para los DAO de cada uno y luego su implementación, vemos sólo el ejemplo de Instalación:

### InstalacionDao.java

```
1 package com.iesvdc.acceso.simplecrud.dao;
2
3 import java.util.List;
4
5 import com.iesvdc.acceso.simplecrud.model.Instalacion;
6
7 public interface InstalacionDao {
8     public boolean create(Instalacion instala);
9     public Instalacion findById(Integer id);
10    public List<Instalacion> findAll();
```

```

11     public List<Instalacion> findByName(String nombre);
12     public boolean update(Instalacion old_al, Instalacion new_al);
13     public boolean update(Integer old_id, Instalacion new_al);
14     public boolean delete(Instalacion instala);
15     public boolean delete(Integer id_al);
16 }

```

### Leer todos/ uno (InstalacionDaoImpl.java)

```

1      @Override
2      public Instalacion findById(Integer id) {
3          Instalacion instala;
4          try {
5              Conexion conexion = new Conexion();
6              Connection conn = conexion.getConnection();
7              String sql = "SELECT * FROM instalacion WHERE id=?";
8              PreparedStatement pstmt = conn.prepareStatement(sql);
9              pstmt.setInt(1, id);
10             System.out.println("\nID:: " + id + "\n");
11             ResultSet rs = pstmt.executeQuery();
12             rs.next();
13             instala = new Instalacion(rs.getInt("id"),
14                         rs.getString("nombre"));
15             conn.close();
16         } catch (SQLException ex) {
17             instala = null;
18         }
19         return instala;
20     }
21
22     @Override
23     public List<Instalacion> findAll() {
24         Instalacion instala;
25         List<Instalacion> li_ins = new ArrayList<Instalacion>();
26         try {
27             Conexion conexion = new Conexion();
28             Connection conn = conexion.getConnection();
29
30             String sql = "SELECT * FROM instalacion";
31
32             PreparedStatement pstmt = conn.prepareStatement(sql);
33
34             ResultSet rs = pstmt.executeQuery();
// recorro el resultset mientras tengo datos

```

```

35         while (rs.next()) {
36             instala = new Instalacion(rs.getInt("id"),
37                                       rs.getString("nombre"));
38             li_ins.add(instala);
39         }
40         // cerramos la conexión
41         conn.close();
42     } catch (SQLException ex) {
43         System.out.println("ERROR" + ex.getMessage());
44         li_ins = null;
45     }
46     return li_ins;
}

```

### Crear (InstalacionDaoImpl.java)

```

1  @Override
2  public boolean create(Instalacion instala) {
3      boolean exito = true;
4      try {
5          Conexion conexion = new Conexion();
6          Connection conn = conexion.getConnection();
7          String sql = "INSERT INTO instalacion VALUES (NULL,?,?)";
8          PreparedStatement pstmt = conn.prepareStatement(sql);
9          pstmt.setInt(1, instala.getId());
10         pstmt.setString(2, instala.getName());
11         pstmt.executeUpdate();
12         conn.close();
13     } catch (SQLException ex) {
14         System.out.println("ERROR: " + ex.getMessage());
15         exito = false;
16     }
17     return exito;
18 }

```

### Actualizar (InstalacionDaoImpl.java)

```

1  /**
2  * Este método actualiza uninstalaummo en la BBDD
3  *
4  * @param old_al El objeto que contiene los datos antiguos
5  *               delinstalaummo

```

```

5      * @param new_al El objeto que contiene los datos nuevos
6          delinstalaummo
7      * @return true si se lleva a cabo correctamente <br>
8          false si no se actualiza nada (error de conexión, no
9          estaba
10         elinstalaummo en la BBDD...) <br>
11     */
12     @Override
13     public boolean update(Instalacion old_al, Instalacion new_al) {
14
15         return update(old_al.getId(), new_al);
16     }
17
18 /**
19     * Este método actualiza una instalación en la BBDD
20     *
21     * @param old_id El id antiguo delinstalaummo
22     * @param new_al El objeto que contieneinstalaummo
23     *               actualizado
24     * @return true si se lleva a cabo correctamente <br>
25     *         false si no se actualiza nada (error de conexión, no
26     *               estaba
27     *               elinstalaummo en la BBDD...) <br>
28     */
29     @Override
30     public boolean update(Integer old_id, Instalacion new_al) {
31         boolean exito = true;
32         try {
33             Conexion conexion = new Conexion();
34             Connection conn = conexion.getConnection();
35             String sql = "UPDATE instalacion SET id=?, nombre=?"
36                         + " WHERE id=?";
37             PreparedStatement pstmt = conn.prepareStatement(sql);
38             pstmt.setInt(3, old_id);
39             pstmt.setInt(1, new_al.getId());
40             pstmt.setString(2, new_al.getName());
41             if (pstmt.executeUpdate() == 0) {
42                 exito = false;
43             }
44         } catch (SQLException ex) {
45             exito = false;
46         }
47         return exito;
48     }

```

### Borrar (InstalacionDaoImpl.java)

```
1  /**
2   * Este método borra de la BBDD el Alumno cuyos datos coinciden
3   * con los de el
4   * objeto que se le pasa como parámetro
5   *
6   * @param instalacion alumno a borrar
7   * @return true si borra uninstalaumno <br>
8   *         false si elinstalaumno no existe o no se puede
9   *         conectar a la BBDD
10  *         <br>
11  */
12 @Override
13 public boolean delete(Instalacion instala) {
14     return delete(instala.getId());
15 }
16
17 @Override
18 public boolean delete(Integer id_al) {
19     boolean exito = true;
20     try {
21         Conexion conexion = new Conexion();
22         Connection conn = conexion.getConnection();
23         String sql = "DELETE FROM instalacion WHERE id=?";
24         PreparedStatement pstmt = conn.prepareStatement(sql);
25         pstmt.setInt(1, id_al);
26         if (pstmt.executeUpdate() == 0) {
27             exito = false;
28         }
29         conn.close();
30     } catch (SQLException ex) {
31         exito = false;
32     }
33     return exito;
34 }
```

## Añadiendo seguridad a la aplicación

Dada la similitud de este sistema con otros sistemas de seguridad de diferentes frameworks y tecnologías, nosotros vamos a recurrir a un filtro para asegurar partes de nuestra Web.

Un filtro nos permite injectar precondiciones a una petición Web (HTTP GET, POST, PUT, DELETE....) o bien postcondiciones, es decir, antes o después de llamar al servlet que despacha la petición del verbo HTTP, se ejecutaría también el método indicado en el filtro.

Esto nos puede ayudar, por ejemplo, para dar seguridad a nuestras aplicaciones. Si tenemos un formulario que haga POST a un servicio de login, desde el servlet que se encarga del servicio, podríamos crear la sesión o cookie que el filtro después comprobará para ver si estamos *loggeados*.

## El servicio de gestión de sesión

Nuestro servicio consta de un formulario de login que hace un POST al servlet que crea la sesión y manda la cookie al cliente para mantener la sesión.

Un filtro intercepta las peticiones a las URLs protegidas y comprueba que hayamos entrado en el sistema con un usuario válido. Aún no estamos usando ACLs (listas de acceso de usuarios).

Para cerrar la sesión un servlet eliminará la cookie.

### Creando la sesión

**Formulario de login:** Hace un POST al servlet que comprueba contra la base de datos si existe el usuario con esa contraseña.

```
1 <div class="container">
2   <div class="jumbotron"><h2>Login</h2></div>
3   <div class="form">
4     <form action="login" method="POST">
5       <input name="username" type="text" placeholder="Nombre
6           de usuario" />
7       <input name="pwd" type="password"
8           placeholder="Contraseña" />
9       <button type="submit">Enviar</button>
10      </form>
11    </div>
12 </div>
```

**Servlet de login:** Recoge los datos del formulario anterior y comprueba contra la base de datos si existe el usuario con esa contraseña.

```
1 /**
2 * Servlet implementation class LoginServlet
```

```

3  */
4 @WebServlet("/login")
5 public class Login extends HttpServlet {
6
7     protected void doPost(HttpServletRequest request,
8         HttpServletResponse response)
9             throws ServletException, IOException {
10
11     Conexion conn = new Conexion();
12     try {
13         Connection conexion = conn.getConnection();
14
15         // get request parameters for userID and password
16         String user = request.getParameter("username");
17         String pwd = request.getParameter("pwd");
18
19         String sql = "SELECT * FROM usuario WHERE username=? AND
20                     password=?";
21         PreparedStatement pstm = conexion.prepareStatement(sql);
22         pstm.setString(1, user);
23         pstm.setString(2, pwd);
24         ResultSet rs = pstm.executeQuery();
25
26         if (rs.next()) {
27             HttpSession session = request.getSession();
28             session.setAttribute("user", user);
29             // setting session to expiry in 30 mins
30             session.setMaxInactiveInterval(30 * 60);
31             Cookie userName = new Cookie("ges_res.user", user);
32             userName.setMaxAge(30 * 60);
33             response.addCookie(userName);
34             response.sendRedirect("index.jsp");
35         } else {
36             RequestDispatcher rd =
37                 getServletContext().getRequestDispatcher("/login.jsp");
38             PrintWriter out = response.getWriter();
39             // out.println("<font color=red>Either user name or password is
40             // wrong."+"</font>");
41             rd.include(request, response);
42         }
43
44         conexion.close();
45     } catch (SQLException e) {
46         // response.sendRedirect("login.jsp");
47         response.getWriter().print(e.getLocalizedMessage());
48     }
49 }

```

```
46    }
47
48 }
```

## Identificando sesión

**Mapeado del Filtro:** web.xml: Indicamos las URLs que están protegidas (son interceptadas) por el filtro Java.

```
1 <filter>
2     <filter-name>UserFilter</filter-name>
3     <filter-class>
4         com.iesvdc.acceso.simplecrud.controller.AuthenticationFilter
5     </filter-class>
6 </filter>
7
8 <filter-mapping>
9     <filter-name>UserFilter</filter-name>
10    <url-pattern>/user/*</url-pattern>
11    <url-pattern>/installation/*</url-pattern>
12    <url-pattern>/privado/*</url-pattern>
13 </filter-mapping>
```

**Filter Java:** Implementación del filtro.

```
1 public class AuthenticationFilter implements javax.servlet.Filter {
2     public static final String AUTHENTICATION_HEADER =
3             "Authorization";
4
5     @Override
6     public void doFilter(ServletRequest request, ServletResponse
7             response,
8             FilterChain filter) throws IOException, ServletException
9     {
10        if (request instanceof HttpServletRequest) {
11            HttpServletRequest httpServletRequest =
12                (HttpServletRequest) request;
13
14            Cookie loginCookie = null;
15            Cookie[] cookies = httpServletRequest.getCookies();
16            if (cookies != null) {
17                for (Cookie cookie : cookies) {
18                    if (cookie.getName().equals("ges_res.user")) {
19                        loginCookie = cookie;
20                    }
21                }
22            }
23        }
24    }
25}
```

```

16             break;
17         }
18     }
19 }
20 if (loginCookie != null) {
21     filter.doFilter(request, response);
22 } else {
23     // ((HttpServletResponse)response.sendRedirect("login.jsp"));
24 if (response instanceof HttpServletResponse) {
25     HttpServletResponse httpServletResponse =
26         (HttpServletResponse) response;
27     httpServletResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
28     httpServletResponse.sendRedirect("/login.jsp");
29 }
30 }
31 }
32 }
33
34 @Override
35 public void destroy() {
36 }
37
38 @Override
39 public void init(FilterConfig arg0) throws ServletException {
40 }
41 }

```

## Cerrando sesión

**Servlet Logout:** Elimina la cookie.

```

1 @WebServlet("/logout")
2 public class Logout extends HttpServlet {
3
4     protected void doPost(HttpServletRequest request,
5         HttpServletResponse response)
6         throws ServletException, IOException {
7     Cookie loginCookie = null;
8     Cookie[] cookies = request.getCookies();
9     if (cookies != null) {
10         for (Cookie cookie : cookies) {
11             if (cookie.getName().equals("ges_res.user")) {
12                 loginCookie = cookie;
13             }
14         }
15         if (loginCookie != null) {
16             response.addCookie(loginCookie);
17         }
18     }
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```

```

12             break;
13         }
14     }
15 }
16 if (loginCookie != null) {
17     loginCookie.setMaxAge(0);
18     response.addCookie(loginCookie);
19 }
20 response.sendRedirect("login.jsp");
21 }
22
23 protected void doGet(HttpServletRequest request,
24     HttpServletResponse response)
25     throws ServletException, IOException {
26     this.doPost(request, response);
27 }
28 }
```

## WS: Servicios REST

Hasta ahora hemos visto un enfoque tradicional (formularios con métodos GET y POST) y un enfoque híbrido (métodos GET, POST, PUT y DELETE) con algo de JavaScript.

Otro enfoque a la hora de diseñar aplicaciones para la Web es la filosofía SAP (Single Application Page), donde usamos todas las características del modelo de cajas de HTML5.

### Single Application Page

Se trata de tener una aplicación HTML5+JS donde existen varias cajas ocultas que contienen las diferentes vistas de la aplicación. En cada momento vamos cambiando de una a otra según un menú principal que hace de controlador para activar/desactivar vistas.

La aplicación se comunica con un servicio REST con verbos HTTP (ej. GET/-POST/PUT/DELETE).

Vamos a implementar las siguientes rutas y verbos HTTP:

Descripción	Verbo HTTP	ruta
Listar todas las instalaciones	GET	/api/installacion
Ver el detalle de una instalación	GET	/api/installacion/{id}
Crear una nueva instalación	POST	/api/installacion
Borrar una instalación	DELETE	/api/installacion/{id}
Buscar una instalación por nombre	GET	/api/installacion/nombre/{nombre}

Veremos este apartado con más detalle en la siguiente clase.

## Creación de un WebService con Tomcat y Jersey

Jersey es un Servlet (aplicación) genérica a la que le indicamos el paquete o clase que queremos exponer a nuestro servicio y **automágicamente** se encarga de hacerlo.

### Añadiendo las dependencias al proyecto

Para poder usar Jersey, primero hemos de añadir las dependencias necesarias en nuestro pom.xml.

Necesitamos JAXB como marshaller/unmarshaller (artefactos jaxb\*), esto nos convierte de objeto plano Java a una interpretación XML (en memoria).

Jersey dispone de tres partes:

1. Conector JAXB: Interpreta las representaciones internas JAXB (marshaller/unmarshaller)
2. Generador de JSON: Convierte las representaciones internas de/hacia JSON
3. Servlet: Es el servicio propiamente dicho (habrá que añadir una entrada en el web.xml a tal efecto).

```

1 <dependency>
2   <groupId>javax.xml.bind</groupId>
3   <artifactId>jaxb-api</artifactId>
4   <version>2.3.0</version>
5 </dependency>
6
7 <dependency>
8   <groupId>com.sun.xml.bind</groupId>
```

```

9      <artifactId>jaxb-core</artifactId>
10     <version>2.3.0</version>
11   </dependency>
12
13   <dependency>
14     <groupId>com.sun.xml.bind</groupId>
15     <artifactId>jaxb-impl</artifactId>
16     <version>2.3.0</version>
17   </dependency>
18
19
20   <dependency>
21     <groupId>javax.activation</groupId>
22     <artifactId>activation</artifactId>
23     <version>1.1.1</version>
24   </dependency>
25
26   <dependency>
27     <groupId>org.glassfish.jersey.media</groupId>
28     <artifactId>jersey-media-jaxb</artifactId>
29     <version>2.25.1</version>
30   </dependency>
31   <dependency>
32     <groupId>org.glassfish.jersey.media</groupId>
33     <artifactId>jersey-media-json-jackson</artifactId>
34     <version>2.25.1</version>
35   </dependency>
36   <dependency>
37     <groupId>org.glassfish.jersey.containers</groupId>
38     <artifactId>jersey-container-servlet-core</artifactId>
39     <version>2.25.1</version>
40   </dependency>

```

## Creando el POJO

Para que podamos trabajar con JAXB, necesitamos trabajar con objetos Java sencillos. Ejemplo:

```

1 package com.iesvdc.acceso.simplecrud.model;
2
3 public class Instalacion {
4
5     private int id;
6     private String name;

```

```

7
8     public Instalacion(){}
9
10    public Instalacion(String name) {
11        this.name = name;
12    }
13
14    public Instalacion(int id, String name) {
15        this.id = id;
16        this.name = name;
17    }
18
19    public int getId() {
20        return id;
21    }
22
23    public void setId(int id) {
24        this.id = id;
25    }
26
27    public String getName() {
28        return name;
29    }
30
31    public void setName(String name) {
32        this.name = name;
33    }
34
35 }
```

## Patrón DAO

Usaremos el patrón DAO (Data Access Object) para crear una clase que se encargue de salvar el *desfase objeto-relacional*. Ejemplo:

```

1 /*
2  * To change this license header, choose License Headers in Project
3  * Properties.
4  * To change this template file, choose Tools / Templates
5  * and open the template in the editor.
6 */
6 package com.iesvdc.acceso.simplecrud.model;
7
8 import java.sql.Connection;
```

```

9 import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.util.ArrayList;
13 import java.util.List;
14
15 import javax.naming.Context;
16 import javax.naming.InitialContext;
17 import javax.sql.DataSource;
18
19 /**
20 *
21 * @author juangu
22 */
23 public class InstalacionDAO {
24     // CRUD, findAll, finById, count
25     Connection conn;
26
27     public InstalacionDAO(){
28         conn = new Conexion().getConnection();
29     }
30
31     public InstalacionDAO(Connection conexion){
32         this.conn=conexion;
33     }
34
35     public boolean create(Instalacion instala){
36         boolean exito=true;
37         try {
38             // Conexion conexion = new Conexion();
39             // Connection conn = conexion.getConnection();
40             String sql =
41                 "INSERT INTO instalacion VALUES (NULL,?,?)";
42             PreparedStatement pstmt = conn.prepareStatement(sql);
43             pstmt.setInt(1, instala.getId());
44             pstmt.setString(2, instala.getName());
45             pstmt.executeUpdate();
46             // conn.close();
47         } catch (SQLException ex) {
48             System.out.println("ERROR: "+ex.getMessage());
49             exito = false;
50         }
51         return exito;
52     }
53
54     public Instalacion finById(Integer id){

```

```

55     Instalacion instala;
56     try {
57         String sql =
58             "SELECT * FROM instalacion WHERE id=?";
59         PreparedStatement pstmt = conn.prepareStatement(sql);
60         pstmt.setInt(1, id);
61         System.out.println("\nID:: "+id+"\n");
62         ResultSet rs = pstmt.executeQuery();
63         rs.next();
64         instala = new Instalacion(
65             rs.getInt("id"),
66             rs.getString("nombre"));
67         // conn.close();
68     } catch (SQLException ex) {
69         instala = null;
70     }
71     return instala;
72 }
73
74 public List<Instalacion> findAll() {
75     Instalacion instala;
76     List<Instalacion> li_ins = new ArrayList();
77     try {
78
79         String sql = "SELECT * FROM instalacion";
80
81         PreparedStatement pstmt = conn.prepareStatement(sql);
82
83         ResultSet rs = pstmt.executeQuery();
84         // recorro el resultset mientras tengo datos
85         while (rs.next()){
86             instala = new Instalacion(
87                 rs.getInt("id"),
88                 rs.getString("nombre"));
89             li_ins.add(instala);
90         }
91         // cerramos la conexión
92         // conn.close();
93     } catch (SQLException ex) {
94         System.out.println("ERROR"+ ex.getMessage());
95         li_ins = null;
96     }
97     return li_ins;
98 }
99
100

```

```

101 /**
102 * Este método busca Instalaciones en la BBDD por nombre:
103 * @param nombre
104 * El nombre a buscar
105 * @return
106 * Devuelve:
107 * null: si hay instalaciones en la BBDD...). <br>
108 * ArrayList vacío (length == 0): si no hay nadie con ese nombre. <br>
109 * ArrayList con Instalaciones: si hay instalaciones con ese nombre.<br>
110 */
111 public List<Instalacion> findByNombre(String nombre){
112     Instalacion instala;
113     List<Instalacion> li_ins = new ArrayList();
114     try {
115         // conectamos a la BBDD
116         Conexion conexion = new Conexion();
117         Connection conn = conexion.getConnection();
118         // esta es la cadena SQL de consulta
119         String sql = "SELECT * FROM instalacion WHERE nombre=?";
120         // usamos este objeto porque es más seguro
121         PreparedStatement pstmt = conn.prepareStatement(sql);
122         pstmt.setString(1, nombre);
123         // ejecutar la consulta contra la base de datos y
124         // devuelve el resultado en el ResultSet (parecido a
125         // un Array con iterador
126         ResultSet rs = pstmt.executeQuery();
127         // corro el resultSet mientras tengo datos
128         while (rs.next()){
129             instala = new Instalacion(
130                 rs.getInt("id"),
131                 rs.getString("nombre"));
132             li_ins.add(instala);
133         }
134         // cerramos la conexión
135         conn.close();
136     } catch (SQLException ex) {
137         System.out.println("ERROR"+ ex.getMessage());
138         li_ins = null;
139     }
140     return li_ins;
141 }
142
143

```

```

144
145
146 /**
147 * Este método actualiza uninstalaumno en la BBDD
148 * @param old_al
149 * El objeto que contiene los datos antiguos delinstalaumno
150 * @param new_al
151 * El objeto que contiene los datos nuevos delinstalaumno
152 * @return
153 * true si se lleva a cabo correctamente <br>
154 * false si no se actualiza nada (error de conexión, no
155 * estaba elinstalaumno en la BBDD...) <br>
156 */
157 public boolean update(Instalacion old_al, Instalacion new_al) {
158
159     return update(old_al.getId(),new_al);
160 }
161
162 /**
163 * Este método actualiza una instalación en la BBDD
164 * @param old_id
165 * El id antiguo delinstalaumno
166 * @param new_al
167 * El objeto que contiene instalacion actualizado
168 * @return
169 * true si se lleva a cabo correctamente <br>
170 * false si no se actualiza nada (error de conexión, no
171 * estaba elinstalaumno en la BBDD...) <br>
172 */
173 public boolean update(Integer old_id, Instalacion new_al) {
174     boolean exito=true;
175     try {
176         Conexion conexion = new Conexion();
177         Connection conn = conexion.getConnection();
178         String sql =
179             "UPDATE instalacion SET id=?, nombre=? WHERE id=?";
180         PreparedStatement pstmt = conn.prepareStatement(sql);
181         pstmt.setInt(3, old_id);
182         pstmt.setInt(1, new_al.getId());
183         pstmt.setString(2, new_al.getName());
184         if (pstmt.executeUpdate()==0) {
185             exito = false;
186         }
187         conn.close();
188     } catch (SQLException ex) {
189         exito = false;

```

```

190         }
191         return exito;
192     }
193
194    /**
195     * Este método borra de la BBDD el Instalacion cuyos datos
196     * coinciden con los de el objeto que se le pasa como
197     * parámetro
198     * @param instalacion a borrar
199     * @return
200     * true si borra un instalacion <br>
201     * false si el instalacion no existe o no se puede conectar a la
202     * BBDD <br>
203     */
204    public boolean delete(Instalacion instalacion){
205        return delete(instalacion.getId());
206    }
207
208    public boolean delete(Integer id_instalacion){
209        boolean exito=true;
210        try {
211            Conexion conexion = new Conexion();
212            Connection conn = conexion.getConnection();
213            String sql = "DELETE FROM instalacion WHERE id=?";
214            PreparedStatement pstmt = conn.prepareStatement(sql);
215            pstmt.setInt(1, id_instalacion);
216            if (pstmt.executeUpdate()==0) {
217                exito = false;
218            }
219            conn.close();
220        } catch (SQLException ex) {
221            exito = false;
222        }
223        return exito;
224    }
225 }
```

## El servicio web (WS)

Primero creamos el recurso que usará Jersey:

```

1 package com.iesvdc.acceso.simplecrud.controller.service;
2
```

```

3 import com.iesvdc.acceso.simplecrud.modelo.*;
4
5 import java.util.ArrayList;
6 import java.util.List;
7 import java.util.logging.Logger;
8 import javax.ws.rs.Consumes;
9 import javax.ws.rs.GET;
10 import javax.ws.rs.POST;
11 import javax.ws.rs.PUT;
12 import javax.ws.rs.DELETE;
13 import javax.ws.rs.Path;
14 import javax.ws.rs.PathParam;
15 import javax.ws.rs.Produces;
16 import javax.ws.rs.core.MediaType;
17 import javax.ws.rs.core.Response;
18
19 /**
20 *
21 * @author Juangu <jgutierrez at iesvirgendelcarmen.coms>
22 */
23 @Path("/")
24 public class InstalacionResource {
25
26     @GET
27     @Path("instalacion")
28     @Produces({MediaType.APPLICATION_JSON,
29                 MediaType.APPLICATION_XML})
30     public List<Instalacion> getInstalaciones() {
31         InstalacionDAO al_dao = new InstalacionDAO();
32         List<Instalacion> list_al;
33         try {
34             list_al = al_dao.findAll();
35         } catch (Exception ex) {
36             list_al = new ArrayList<>();
37             Logger.getLogger(ex.getLocalizedMessage());
38         }
39         return list_al;
40     }
41
42     @GET
43     @Path("instalacion/{id}")
44     @Produces({MediaType.APPLICATION_JSON,
45                 MediaType.APPLICATION_XML})
46     public Instalacion getInstalacionById(@PathParam("id") String
47                                         id) {
48         InstalacionDAO al_dao = new InstalacionDAO();

```

```

46     Instalacion al;
47     try {
48         al = al_dao.findById(Integer.parseInt(id));
49     } catch (Exception ex) {
50         al = new Instalacion(-1, "Error");
51         Logger.getLogger(ex.getLocalizedMessage());
52     }
53     return al;
54 }
55
56
57 @GET
58 @Path("instalacion/nombre/{nombre}")
59 @Produces({MediaType.APPLICATION_JSON,
60             MediaType.APPLICATION_XML})
61 public List<Instalacion>
62     getInstalacionByNombre(@PathParam("nombre") String nombre) {
63     InstalacionDAO al_dao = new InstalacionDAO();
64     List<Instalacion> list_al;
65     try {
66         list_al = al_dao.findByNombre(nombre);
67     } catch (Exception ex) {
68         list_al = new ArrayList<>();
69         Logger.getLogger(ex.getLocalizedMessage());
70     }
71     return list_al;
72 }
73
74 @PUT
75 @Path("instalacion/{id}")
76 @Consumes({MediaType.APPLICATION_JSON,
77             MediaType.APPLICATION_XML})
78 public void updateInstalacion(@PathParam("id") Integer id,
79     Instalacion al) {
80     InstalacionDAO al_dao = new InstalacionDAO();
81     try {
82         al_dao.update(id, al);
83     } catch (Exception ex) {
84         Logger.getLogger(ex.getLocalizedMessage());
85     }
86 }
87
88
89 @POST

```

```

87     @Consumes({MediaType.APPLICATION_JSON,
88                 MediaType.APPLICATION_XML})
89     @Produces({MediaType.APPLICATION_JSON,
90                 MediaType.APPLICATION_XML})
91     @Path("alumno")
92     public Response createInstalacion(Instalacion al) {
93         InstalacionDAO al_dao = new InstalacionDAO();
94         try {
95             al_dao.create(al);
96         } catch (Exception ex) {
97             Logger.getLogger(ex.getLocalizedMessage());
98             return Response.status(400).entity(al).build();
99         }
100        return Response.status(200).entity(al).build();
101    }
102    @DELETE
103    @Path("instalacion/{id}")
104    public void deleteInstalacion(@PathParam("id") Integer id) {
105        InstalacionDAO al_dao = new InstalacionDAO();
106        try {
107            al_dao.delete(id);
108        } catch (Exception ex) {
109            Logger.getLogger(ex.getLocalizedMessage());
110        }
111    }

```

Ahora hay que modificar el archivo **web.xml** para dar de alta el servlet Jersey en la ruta “/rest”:

```

1 <servlet>
2     <servlet-name>jersey-servlet</servlet-name>
3     <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
4     <init-param>
5         <param-name>jersey.config.server.provider.packages</param-name>
6         <param-value>com.iesvdc.acceso.simplecrud.controller.service</param-value>
7     </init-param>
8     <init-param>
9         <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
10        <param-value>true</param-value>
11    </init-param>
12    <load-on-startup>1</load-on-startup>
13 </servlet>
14 <servlet-mapping>
15     <servlet-name>jersey-servlet</servlet-name>

```

```
16      <url-pattern>/rest/*</url-pattern>
17    </servlet-mapping>
```

## Probando el servicio

Hasta que tengamos el frontend, podemos hacer nuestros tests con la extensión de Mozilla Firefox. “REST Client”.

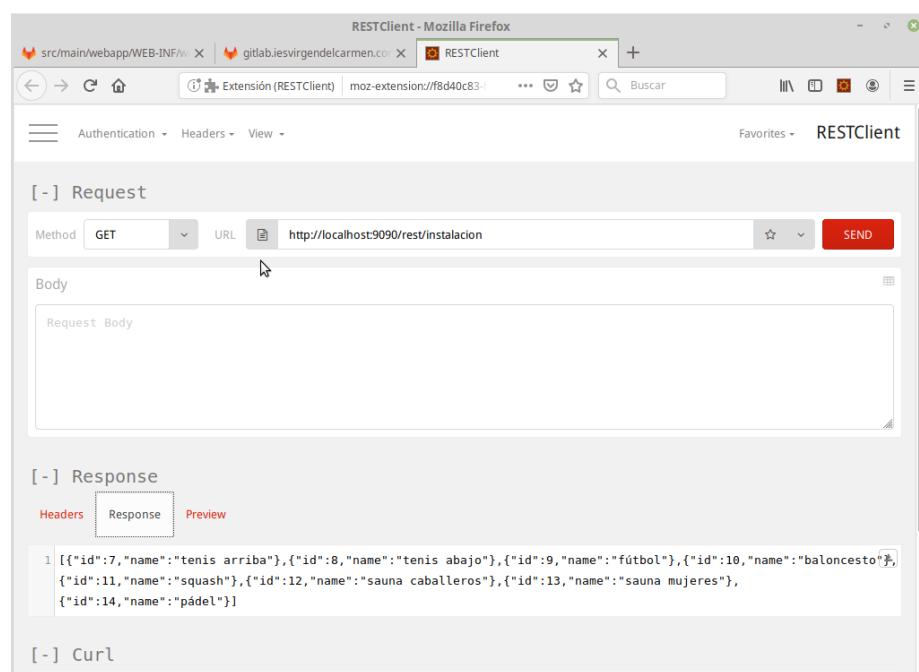


Figure 2: Extensión RESTClient de firefox

## Aplicación híbrida

Recordamos que se trata de tener una aplicación HTML5+JS donde existen varias cajas ocultas que contienen las diferentes vistas de la aplicación. En cada momento vamos cambiando de una a otra según un menú principal que hace de controlador para activar/desactivar vistas.

La aplicación se comunica con el servicio REST con verbos HTTP (ej. GET/-/POST/PUT/DELETE).

Ya tenemos implementados los siguientes *end-points* en el **back-end**:

Descripción	Verbo HTTP	ruta
Listar todas las instalaciones	GET	/api/installacion
Ver el detalle de una instalación	GET	/api/installacion/{id}
Crear una nueva instalación	POST	/api/installacion
Borrar una instalación	DELETE	/api/installacion/{id}
Buscar una instalación por nombre	GET	/api/installacion/nombre/{nombre}

Ahora vamos a preparar una APP híbrida como **front-end**, que con AJAX haga esas consultas y muestre los resultados.

## Modificando el backend para que soporte peticiones de otro origen

Para resolver el problema del CORS deberemos crear un filtro que intercepte peticiones de origen cruzado y las permita sólo para los servicios visibles a nuestra aplicación.

Ejemplo de filtro CORS Java:

```
1 import java.io.IOException;
2
3 import javax.servlet.Filter;
4 import javax.servlet.FilterChain;
5 import javax.servlet.FilterConfig;
6 import javax.servlet.ServletException;
7 import javax.servlet.ServletRequest;
8 import javax.servlet.ServletResponse;
9 import javax.servlet.http.HttpServletResponse;
10
11 public class CORSFilter implements Filter {
12
13     public void doFilter(
14         ServletRequest req,
15         ServletResponse res,
16         FilterChain chain) throws IOException, ServletException {
17
18     HttpServletResponse response = (HttpServletResponse) res;
```

```

19     response.setHeader("Access-Control-Allow-Origin", "*");
20     response.setHeader("Access-Control-Allow-Methods", "POST,
21         GET, PUT, DELETE");
22     response.setHeader("Access-Control-Max-Age", "3600");
23     response.setHeader("Access-Control-Allow-Headers",
24         "Origin, x-requested-with, Content-Type, Accept");
25     chain.doFilter(req, res);
26 }
27
28     public void init(FilterConfig filterConfig) {}
29
30     public void destroy() {}
31 }
```

Recuerda modificar el fichero **web.xml** para que la ruta del Web Service tenga permitido el acceso desde orígenes desconocidos:

```

1 <filter>
2     <filter-name>CorsFilter</filter-name>
3     <filter-class>com.iesvdc.acceso.simplecrud.controller.CORSFilter</filter-class>
4 </filter>
5
6 <filter-mapping>
7     <filter-name>CorsFilter</filter-name>
8     <url-pattern>/rest/*</url-pattern>
9 </filter-mapping>
```

## Preparación del entorno

Creamos una carpeta frontend en nuestro proyecto Web. Para crear el proyecto Cordova, ejecutamos los siguientes comandos en la terminal dentro del directorio recién creado:

```

1 $ npm install -g cordova
2 $ cordova create reservas com.iesvdc.dam.acceso ReservAPP
3 $ cd reservas
4 $ cordova platform add browser
5 $ npm install jquery
6 $ npm install popper.js
7 $ npm install bootstrap
```

Ahora, de los directorios node\_modules que se han creado para cada una de las instalaciones. Me creo un script añadir dependencias:

```

1 #!/bin/bash
2
3 mkdir -p www/vendor/js www/vendor/css
4 echo "Añadiendo jQuery"
5 cp node_modules/jquery/dist/jquery.min.js \
6   node_modules/jquery/dist/jquery.min.map www/vendor/js
7 echo "Añadiendo PopperJS"
8 cp node_modules/popper.js/dist/popper.min.js \
9   node_modules/popper.js/dist/popper.min.js.map www/vendor/js
10 echo "Añadiendo Bootstrap"
11 cp node_modules/bootstrap/dist/css/bootstrap.min.css www/vendor/css
12 cp node_modules/bootstrap/dist/js/bootstrap.min.js \
13   node_modules/bootstrap/dist/js/bootstrap.min.js.map www/vendor/js

```

Para lanzar un navegador Web Cordova:

```
1 cordova run browser
```

## Ejemplo de aplicación HTML5/JS

En la carpeta `www` recién creada por `cordova`, vamos a editar el fichero `index.html`. La idea es hacer una SAP (Single Application Page), luego vamos a crear una serie de vistas, que llamaremos “paneles”, que serán cajas ocultas (etiqueta `div`). Con un menú vamos mostrando una u otra vista (panel) según donde vamos pulsando.

Ejemplo de paneles (fichero `index.html`):

```

1 <div class="row">
2   <div class="panel" id="panel_inicio">
3     <h2>Modelo servicio REST</h2>
4   </div>
5   <div class="panel" id="panel_inst_read">
6     <h2>Listado de instalaciones</h2>
7     <div id="panel_inst_list" class="input-group mb-3">
8       Aquí va el listado.
9     </div>
10    </div>
11    <div class="panel" id="panel_inst_update">
12      <h2>Actualización de una instalación</h2>
13      <div class="input-group mb-3">
14        <select class="custom-select" id="select_inst_update">
15          <option>instalacion</option>
16        </select>

```

```

17      <button id="btn_inst_update" type="button" class="btn
18          btn-primary">
19              actualizar</button>
20      </div>
21  </div>
22  <div class="panel" id="panel_inst_delete">
23      <h2>Borrado de instalaciones</h2>
24      <div class="input-group mb-3">
25          <select class="custom-select" id="select_inst_delete">
26              <option>instalacion</option>
27          </select>
28          <button id="btn_inst_delete" type="button" class="btn
29              btn-danger">borrar</button>
30      </div>
31  </div>
32  <div class="panel" id="panel_inst_create">
33      <h2>Alta de instalacion</h2>
34      <div class="input-group mb-3">
35          <input type="text" class="form-control"
36              id="nombre_inst_create" />
37          <button id="btn_inst_create" type="button" class="btn
              btn-success">crear</button>
38      </div>
39  </div>
40 </div>

```

Ahora con JavaScript (y jQuery) es muy fácil ocultar todos los DIV de la clase PANEL y mostrar únicamente la vista o panel que nos interese en cada momento. Bastaría con hacer un:

```

1  $(".panel").hide();
2  $("#id_panel_a_mostrar").show();

```

Fíjate bien en los ID que hemos dado a cada panel y ahora veamos el menú de la WebApp, observa también los ID de cada menú del CRUD:

```

1  <div class="dropdown-menu"
2      aria-labelledby="navbarDropdownMenuLink">
3      <a id="menu_inst_read" class="dropdown-item"
4          href="#">Listado</a>
5      <a id="menu_inst_create" class="dropdown-item"
6          href="#">Alta</a>
7      <a id="menu_inst_update" class="dropdown-item"
8          href="#">Actualización</a>
9      <a id="menu_inst_delete" class="dropdown-item"
10         href="#">Baja</a>
11  </div>

```

¿Has visto cómo cada menu\_inst\_\* tiene su panel panel\_inst\_\*? Esto lo hemos hecho así para ahora con JavaScript conectar el evento de pulsar en un menú con su panel correspondiente (fichero www/js/controller.js):

```
1 $.controller.active_panel = "#panel_inicio";
2
3 $.controller.activate = function (panel_name) {
4     ($.controller.active_panel).hide();
5     $(panel_name).show();
6     $.controller.active_panel = panel_name;
7 };
8
9 $.controller.init = function (panel_inicial) {
10    $('[id^="menu_"]').each(function () {
11        var $this = $(this);
12        var menu_id = $this.attr('id');
13        var panel_id = menu_id.replace('menu_', 'panel_');
14
15        $("#" + menu_id).click(function () {
16            $.controller.activate("#" + panel_id);
17        });
18    });
}
```

Ahora falta darle funcionalidad a cada botón del controlador para terminar el CRUD del servicio. Además de completar el código de cada evento “on click” (fichero www/js/controller.js), posiblemente tengamos que añadir algún panel más al fichero index.html:

```
1   $.controller.init = function (panel_inicial) {
2
3     ....
4
5     $("#btn_inst_create").click(()=>{
6         console.log("btn_inst_create-onClick::TODO:: falta llamar al
7             doPost");
8     });
9
10    $("#btn_inst_update").click(()=>{
11        console.log("btn_inst_update-onClick::TODO:: falta llamar al
12            doGet "+
13                "para poblar el formulario y luego hacer el doPut");
14    });
15
16    $("#btn_inst_delete").click(()=>{
17        console.log("btn_inst_delete-onClick::TODO:: falta llamar al
18            doDelete");
19    });
20}
```

```
17  
18 };
```

Ya está hecho el **READ** del **CRUD de instalación** para ayudarte en la tarea, fíjate cómo se hace el “binding” de evento “pulsar sobre el menú read” y generar una tabla a partir del JSON que pedimos al Webservice.

```
1  $("#menu_inst_read").click(()=>{  
2      console.log("listando instalaciones... ");  
3      $.controller.doGet($.controller.url+"instalacion",  
4          function(datos){  
5              console.log("listando instalaciones: "+datos);  
6              $("#panel_inst_read").empty();  
7              let tabla=$("<table/>");  
8              datos.forEach(instalacion => {  
9                  let fila=$("<tr/>");  
10                 fila.append("<td>"+"Instalación  
11                     "+instalacion.id+"</td>");  
12                 fila.append("<td>"+instalacion.name+"</td>");  
13                 fila.append("<td>"+  
14                     "<span class=\"btn btn-success\">ACTUALIZAR</span>  
15                     "+  
16                     "</td>");  
17                 fila.append("<td><span class=\"btn  
18                     btn-danger\">BORRAR</span> </td>");  
19                 tabla.append(fila);  
});  
$("#panel_inst_read").append(tabla);  
});  
});
```

## Localtunel

Para probar el servicio en Internet:

```
npm install -g localtunnel
```

```
lt -port 9090 # Repaso disparadores
```

La sintaxis para crear un disparador es la siguiente:

```
1 CREATE TRIGGER nombre_disparador  
2   [BEFORE|AFTER]  [INSERT|DELETE|UPDATE|...]  
3   ON nombre_tabla FOR EACH ROW  
4   BEGIN
```

```
5      [cuerpo del disparador]
6  END;
```

### Ejemplo 1: No más de una reserva al día por persona

Sea el dominio del proyecto (gestión de reservas), imaginemos que no queremos delegar sólo en el código de la aplicación que se pueda reservar más de una vez por usuario (si esta condición la incluimos en la base de datos, será más segura, será mucho más difícil que un usuario malintencionado pueda romperla).

La alternativa es crear un disparador que primero compruebe si ya tenemos reservas en el día que vamos a insertar la nueva reserva:

```
1 SELECT COUNT(*) FROM `reserva` WHERE `reserva`.`fecha` = NEW.`fecha`
```

Fíjate en el prefijo “NEW.”, en los disparadores, recuerda que esto sirve para acceder a la nueva tupla que queremos insertar, seguido de un punto y luego el nombre de cada columna para acceder al valor.

Antes de insertar una nueva reserva **BEFORE INSERT**, para cada tupla que vamos a insertar **FOR EACH ROW** comprobamos con un **IF** si ya había más de una reserva ese día para un usuario dado. Si ya la había, no procedemos a la inserción y devolvemos un mensaje de error ‘*sólo se permite una reserva al día*’, con código de error *45001* (este código nos lo devuelve MySQL cuando hagamos la consulta).

```
1 -- Sólo deja una reserva para cada usuario
2 DROP TRIGGER IF EXISTS reserva_diaria;
3
4 DELIMITER $$*
5 CREATE TRIGGER `reserva_diaria`
6 BEFORE INSERT ON `reserva` FOR EACH ROW
7 BEGIN
8     IF (SELECT COUNT(*) FROM `reserva`*
9         WHERE `reserva`.`fecha` = NEW.`fecha`*
10            AND `reserva`.`usuario` = NEW.`usuario`) > 0
11    THEN
12        SIGNAL sqlstate '45001'
13        SET message_text = 'Sólo se permite una reserva al día.';
14    END IF;
15 END;
16 $$*
```

## Ejemplo 2: No podemos reservar con más de dos semanas de antelación

Volvamos al problema inicial, el sistema de gestión de reservas. Si no controlamos un poco las fechas de las reservas, podemos encontrarnos que usuarios malintencionados pueden comprometer la integridad del sistema o situaciones indeseadas. Por tanto deberíamos controlar que:

1. No se pueda borrar o actualizar reservas pasadas:
  1. No se puede borrar reservas ya pasadas. Si se permitiera borrar reservas, podríamos borrar una ya pasada, luego podríamos librarnos de pagar las instalaciones que hemos disfrutado (disparador “ON DELETE”).
  2. Al día siguiente, o al haber pasado la hora de una reserva, si ponen esa misma reserva a otro nombre, puede ocurrir que a final de mes en vez de cobrar al usuario que ha disfrutado de la instalación, se le va a cobrar a otro (disparador “ON UPDATE”).
  2. No se puede reservar en fechas anteriores al día actual. No tiene sentido.
  3. No se permite reservar con más de dos semanas de antelación. En caso contrario, un usuario podría reservar todos los días del año a las 18:00 una instalación, por ejemplo.

Ejemplo de disparador (caso 1.1): Observa cómo no podemos usar ahora “NEW.\*” para acceder a un campo, pues no existiría al borrar, sólo podemos usar “OLD.\*” para ello:

```
1 DROP TRIGGER IF EXISTS reserva_pasado;
2
3 DELIMITER $$
```

4 **CREATE TRIGGER** `reserva\_pasado`  
5   **BEFORE DELETE ON** `reserva` **FOR EACH ROW**  
6   **BEGIN**  
7     **IF** ( OLD.`fecha` < CURDATE() )  
8       **THEN**  
9         **SIGNAL** sqlstate '45004'  
10         **SET** message\_text = 'No se permite eliminar una fecha  
11            pasada.';  
12       **END IF;**  
13   **END;**  
14 \$\$

En este ejemplo surge la pregunta... ¿y si borro el mismo día de la reserva?. Esto ya sería una consulta más complicada, pues habría que mirar que la hora actual es menor que la hora de inicio del horario de la reserva (habría que hacer una consulta más compleja con un JOIN). Puedes intentarlo para ampliar.

Ejemplo de disparador (caso 1.2): No se puede cambiar de nombre (a otra persona) una reserva el día de la misma o cuando ha pasado:

```
1 DROP TRIGGER IF EXISTS reserva_actualizar_pasado;
2
3 DELIMITER $$
```

4 **CREATE TRIGGER** `reserva\_actualizar\_pasado`  
5   **BEFORE UPDATE ON** `reserva` **FOR EACH ROW**  
6   **BEGIN**  
7     **IF** ( OLD.`fecha` <= CURDATE() )  
8       **THEN**  
9         **SIGNAL** sqlstate '45004'  
10         **SET** message\_text = 'No se permite actualizar una reserva ya  
11           o casi pasada.';  
12     **END IF;**  
13   **END;**  
14 **\$\$**

Ejemplo de disparador (casos 2 y 3):

```
1 -- Si machacamos reservas antiguas bajo otro nombre, nos libramos de
  pagar a final de mes...
2 DROP TRIGGER IF EXISTS reserva_semanal;
3
4 DELIMITER $$
```

5 **CREATE TRIGGER** `reserva\_semanal`  
6   **BEFORE INSERT ON** `reserva` **FOR EACH ROW**  
7   **BEGIN**  
8     **IF** ( NEW.`fecha` < CURDATE() )  
9       **THEN**  
10         **SIGNAL** sqlstate '45002'  
11         **SET** message\_text = 'No se permite reservar en una fecha  
12           anterior a la actual.';  
13     **ELSEIF** ( NEW.`fecha` > DATE\_ADD(CURDATE(), INTERVAL 14 DAY) )  
14       **THEN**  
15         **SIGNAL** sqlstate '45003'  
16         **SET** message\_text = 'No se permite reservar con más de dos  
17           semanas de antelación.';  
18     **END IF;**  
19   **END;**  
20 **\$\$**

## Comandos útiles de docker

Listado de contenedores:

```
1 $ docker ps
```

Parar un contenedor:

```
1 $ docker stop [contenedor]
```

Borrar un contenedor:

```
1 $ docker rm [contenedor]
```

Listado de imágenes:

```
1 $ docker image ls
```

Borrar una imagen:

```
1 $ docker rmi [imagen]
```

Listado de volúmenes:

```
1 $ docker volume ls
```

Borrar todos los volúmenes que no estén usados por ningún contenedor:

```
1 $ docker volume prune
```

Borrar un volumen concreto:

```
1 $ docker volume rm [volumen]
```