

Contents

Tema 02: Conectividad con Bases de Datos Relacionales	2
Índice de contenido	3
El problema	3
Preparando el entorno	4
Clonando el proyecto	4
Creando los contenedores para la base de datos	4
Preparando el entorno	5
Creamos la BBDD	6
Creación del proyecto en modo interactivo (MAVEN)	10
Dependencias Maven	10
MySQL	10
Soporte J2EE (Servlets)	10
Soporte para JSLT (para JSP)	11
Soporte JSON	11
Insertando el servidor Tomcat en Maven	11
Servlets	12
Ciclo de vida	12
Inicializar el servlet	12
Interactuar con los clientes	12
Destruir el servlet	12
¿Cómo funciona un servlet?	12
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
Conexión a la base de datos	22
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	33
Conectando a la Base de Datos	34
CRUD básico	35
LEER uno (findOne)	35
LEER todos (findAll)	35
Crear	36
Actualizar	37
Borrar	37

CRUD (patrón DAO)	38
InstalacionDao.java	38
Leer todos/ uno (InstalacionDaoImpl.java)	38
Crear (InstalacionDaoImpl.java)	39
Actualizar (InstalacionDaoImpl.java)	40
Borrar (InstalacionDaoImpl.java)	41
Añadiendo seguridad a la aplicación	42
El servicio de gestión de sesión	42
Creando la sesión	42
Identificando sesión	44
Cerrando sesión	45
WS: Servicios REST	46
Single Application Page	46
Creación de un Webservice con Tomcat y Jersey	47
Añadiendo las dependencias al proyecto	47
Creando el POJO	48
Patrón DAO	49
El servicio web (WS)	54
Probando el servicio	57
Aplicación híbrida	57
Modificando el backend para que soporte peticiones de otro origen	58
Preparación del entorno	59
Ejemplo de aplicación HTML5/JS	60
Verbos HTTP comunicación cliente/servidor	62
CRUD Instalación (front-end)	65
Instalación findAll():	66
Instalación create():	67
Instalación update():	67
Instalación delete():	68
Localtunel	68
Ejemplo 1: No más de una reserva al día por persona	69
Ejemplo 2: No podemos reservar con más de dos semanas de antelación	70
Comandos útiles de docker	71

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

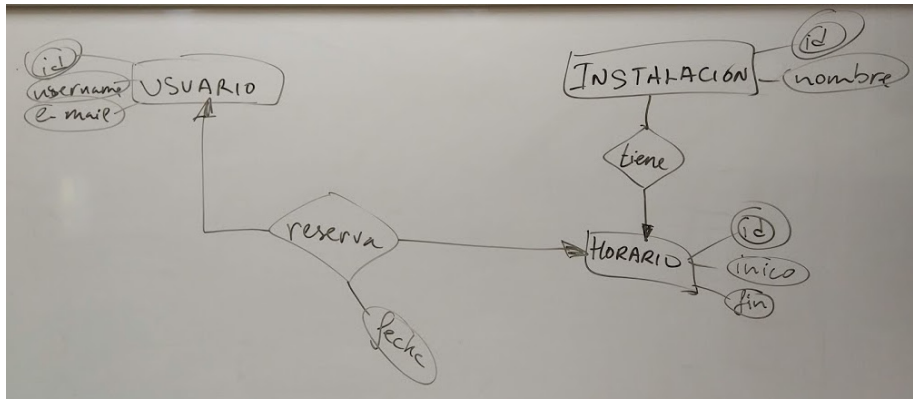


Figure 1: Diagrama ER

4. Los usuarios administradores son los únicos que pueden crear/actualizar/borrar usuarios

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
2 $ 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 *automáticamente* al crear las aplicaciones docker):

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

```

36 `fecha` date NOT NULL,
37 FOREIGN KEY (`usuario`) REFERENCES `usuario` (`id`),
38 FOREIGN KEY (`horario`) REFERENCES `horario` (`id`)
39 ) ENGINE='InnoDB';
40
41 INSERT INTO `usuario` (`id`, `username`, `password`, `email`) VALUES
42 (2, 'pepillo', 'secreto', 'pepe@gmail.com'),
43 (5, 'admin', 'admin', 'admin@correo.com'),
44 (7, 'obijuan', 'starwars', 'darkside@starwars.com'),
45 (13, 'gerente', 'password1234', 'gerencia@vdc.com');
46
47 INSERT INTO `instalacion` (`id`, `nombre`) VALUES
48 (7, 'tenis arriba'),
49 (8, 'tenis césped artificial'),
50 (9, 'fútbol'),
51 (10, 'baloncesto'),
52 (11, 'squash'),
53 (13, 'sauna mujeres'),
54 (14, 'pista de pádel'),
55 (16, 'sauna caballeros');
56
57
58 INSERT INTO `horario` (`id`, `instalacion`, `inicio`, `fin`) VALUES
59 (1, 7, '08:00:00', '09:00:00'),
60 (2, 7, '09:00:00', '10:00:00'),
61 (3, 7, '10:00:00', '11:00:00'),
62 (4, 7, '11:00:00', '12:00:00'),
63 (5, 7, '12:00:00', '13:00:00'),
64 (6, 7, '13:00:00', '14:00:00'),
65 (7, 7, '14:00:00', '15:00:00'),
66 (8, 7, '15:00:00', '16:00:00'),
67 (9, 7, '16:00:00', '17:00:00'),
68 (10, 7, '17:00:00', '18:00:00'),
69 (11, 7, '18:00:00', '19:00:00'),
70 (12, 7, '19:00:00', '20:00:00'),
71 (13, 7, '20:00:00', '21:00:00'),
72 (14, 7, '21:00:00', '22:00:00'),
73 (15, 7, '22:00:00', '23:00:00'),
74 (16, 8, '08:00:00', '09:00:00'),
75 (17, 8, '09:00:00', '10:00:00'),
76 (18, 8, '10:00:00', '11:00:00'),
77 (19, 8, '11:00:00', '12:00:00'),
78 (20, 8, '12:00:00', '13:00:00'),
79 (21, 8, '13:00:00', '14:00:00'),
80 (22, 8, '14:00:00', '15:00:00'),
81 (23, 8, '15:00:00', '16:00:00'),

```

```

82 (24, 8, '16:00:00', '17:00:00'),
83 (25, 8, '17:00:00', '18:00:00'),
84 (26, 8, '18:00:00', '19:00:00'),
85 (27, 8, '19:00:00', '20:00:00'),
86 (28, 8, '20:00:00', '21:00:00'),
87 (29, 8, '21:00:00', '22:00:00'),
88 (30, 8, '22:00:00', '23:00:00'),
89 (31, 9, '08:00:00', '09:30:00'),
90 (32, 9, '09:30:00', '11:00:00'),
91 (33, 9, '11:00:00', '12:30:00'),
92 (34, 9, '12:30:00', '14:00:00'),
93 (35, 9, '14:00:00', '15:30:00'),
94 (36, 9, '15:30:00', '17:00:00'),
95 (37, 9, '17:00:00', '18:30:00'),
96 (38, 9, '18:30:00', '20:00:00'),
97 (39, 9, '20:00:00', '21:30:00'),
98 (40, 9, '21:30:00', '23:00:00'),
99 (41, 9, '23:00:00', '00:30:00'),
100 (42, 10, '08:00:00', '09:00:00'),
101 (43, 10, '09:00:00', '10:00:00'),
102 (44, 10, '10:00:00', '11:00:00'),
103 (45, 10, '11:00:00', '12:00:00'),
104 (46, 10, '12:00:00', '13:00:00'),
105 (47, 10, '13:00:00', '14:00:00'),
106 (48, 10, '14:00:00', '15:00:00'),
107 (49, 10, '15:00:00', '16:00:00'),
108 (50, 10, '16:00:00', '17:00:00'),
109 (51, 10, '17:00:00', '18:00:00'),
110 (52, 10, '18:00:00', '19:00:00'),
111 (53, 10, '19:00:00', '20:00:00'),
112 (54, 10, '20:00:00', '21:00:00'),
113 (55, 10, '21:00:00', '22:00:00'),
114 (56, 10, '22:00:00', '23:00:00'),
115 (57, 11, '08:00:00', '08:45:00'),
116 (58, 11, '08:45:00', '09:30:00'),
117 (59, 11, '09:30:00', '10:15:00'),
118 (60, 11, '10:15:00', '11:00:00'),
119 (61, 11, '11:00:00', '11:45:00'),
120 (62, 11, '11:45:00', '12:30:00'),
121 (63, 11, '12:30:00', '13:15:00'),
122 (64, 11, '13:15:00', '14:00:00'),
123 (65, 11, '14:00:00', '14:45:00'),
124 (66, 11, '14:45:00', '15:30:00'),
125 (67, 11, '15:30:00', '16:15:00'),
126 (68, 11, '16:15:00', '17:00:00'),
127 (69, 11, '17:00:00', '17:45:00'),

```



```

128 (70, 11, '17:45:00', '18:30:00'),
129 (71, 11, '18:30:00', '19:15:00'),
130 (72, 11, '19:15:00', '20:00:00'),
131 (73, 11, '20:00:00', '20:45:00'),
132 (74, 11, '20:45:00', '21:30:00'),
133 (75, 11, '21:30:00', '22:15:00'),
134 (103, 13, '09:00:00', '09:30:00'),
135 (104, 13, '09:30:00', '10:00:00'),
136 (105, 13, '10:00:00', '10:30:00'),
137 (106, 13, '10:30:00', '11:00:00'),
138 (107, 13, '11:00:00', '11:30:00'),
139 (108, 13, '11:30:00', '12:00:00'),
140 (109, 13, '12:00:00', '12:30:00'),
141 (110, 13, '12:30:00', '13:00:00'),
142 (111, 13, '13:00:00', '13:30:00'),
143 (112, 13, '13:30:00', '14:00:00'),
144 (113, 13, '14:00:00', '14:30:00'),
145 (114, 13, '14:30:00', '15:00:00'),
146 (115, 13, '15:00:00', '15:30:00'),
147 (116, 13, '15:30:00', '16:00:00'),
148 (117, 13, '16:00:00', '16:30:00'),
149 (118, 13, '16:30:00', '17:00:00'),
150 (119, 13, '17:00:00', '17:30:00'),
151 (120, 13, '17:30:00', '18:00:00'),
152 (121, 13, '18:00:00', '18:30:00'),
153 (122, 13, '18:30:00', '19:00:00'),
154 (123, 13, '19:00:00', '19:30:00'),
155 (124, 13, '19:30:00', '20:00:00'),
156 (125, 13, '20:00:00', '20:30:00'),
157 (126, 13, '20:30:00', '21:00:00'),
158 (127, 13, '21:00:00', '21:30:00'),
159 (128, 13, '21:30:00', '22:00:00'),
160 (129, 13, '22:00:00', '22:30:00'),
161 (130, 14, '08:00:00', '09:30:00'),
162 (131, 14, '09:30:00', '11:00:00'),
163 (132, 14, '11:00:00', '12:30:00'),
164 (133, 14, '12:30:00', '14:00:00'),
165 (134, 14, '14:00:00', '15:30:00'),
166 (135, 14, '15:30:00', '17:00:00'),
167 (136, 14, '17:00:00', '18:30:00'),
168 (137, 14, '18:30:00', '20:00:00'),
169 (138, 14, '20:00:00', '21:30:00'),
170 (139, 14, '21:30:00', '23:00:00'),
171 (140, 14, '23:00:00', '00:30:00');
172
173

```

```

174 INSERT INTO `reserva` (`id`, `usuario`, `horario`, `fecha`) VALUES
175 (1, 2, 130, '2019-10-12'),
176 (2, 2, 130, '2019-10-13'),
177 (4, 7, 120, '2019-11-11'),
178 (5, 7, 130, '2019-11-21');

```

Creación del proyecto en modo interactivo (MAVEN)

Si has clonado este repositorio, no es necesario hacer este paso, solamente 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/unmarshallig 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. delete

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.

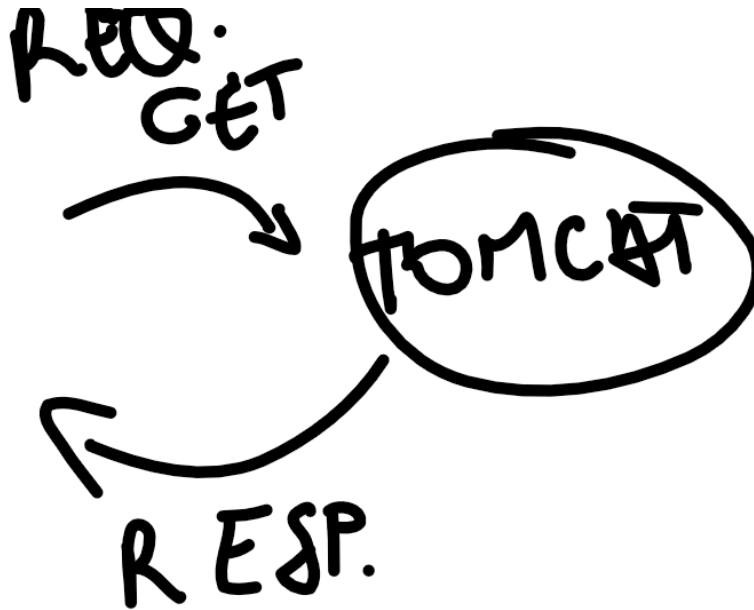
En el código de la clase es el **método destroy**.

¿Cómo funciona un servlet?

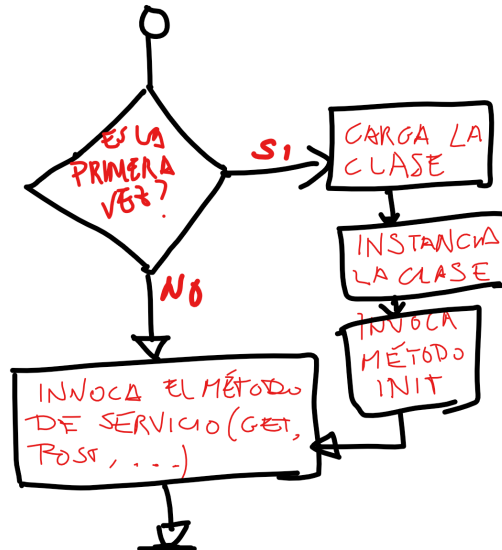
Hasta ahora hemos creado aplicaciones J2SE, es decir aplicaciones para ejecutar en el equipo local.

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.



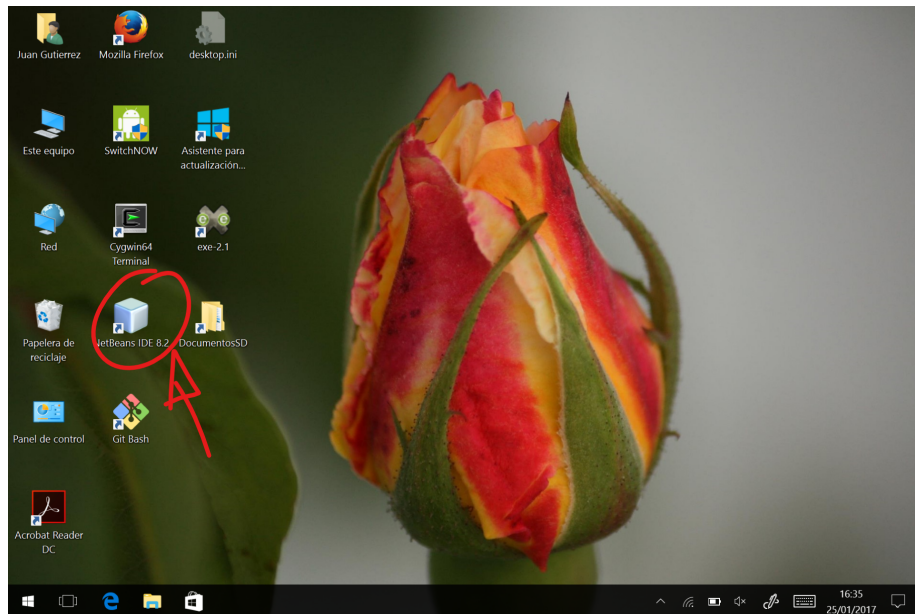
La primera vez que se invoque la ruta de nuestro servlet, éste será cargado e inicializado. Las siguientes veces ya estará en memoria y un hilo de ejecución invocará el método correspondiente.

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 HttpServletRequest object was last modified.
init	Initializes the servlet.
destroy	Finalizes the servlet.
getServletInfo	Provides information regarding the servlet.

Preparando el entorno

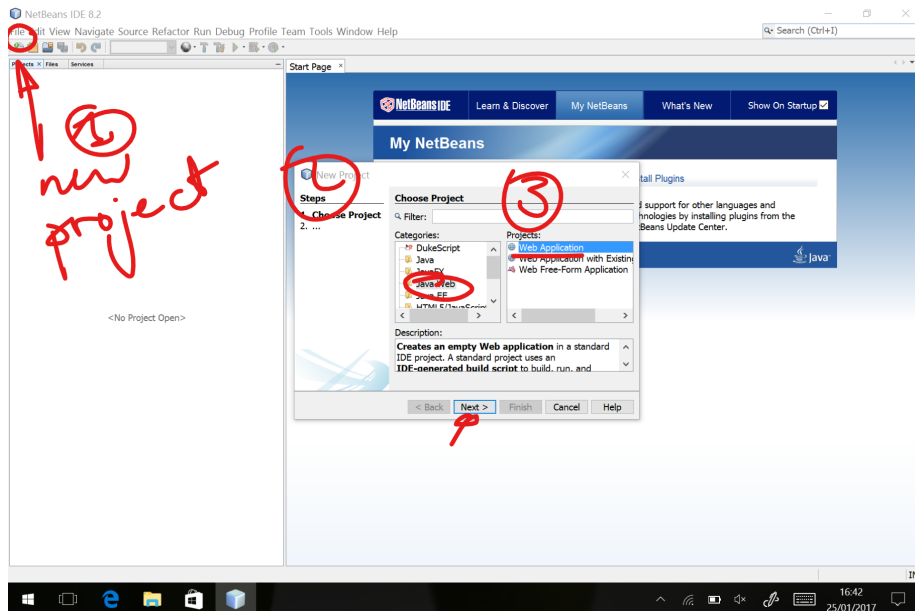
Para crear el proyecto vamos a usar Netbeans IDE y el servidor de aplicaciones J2EE Tomcat.

Partimos que ya hemos descargado e instalado NetBeans.

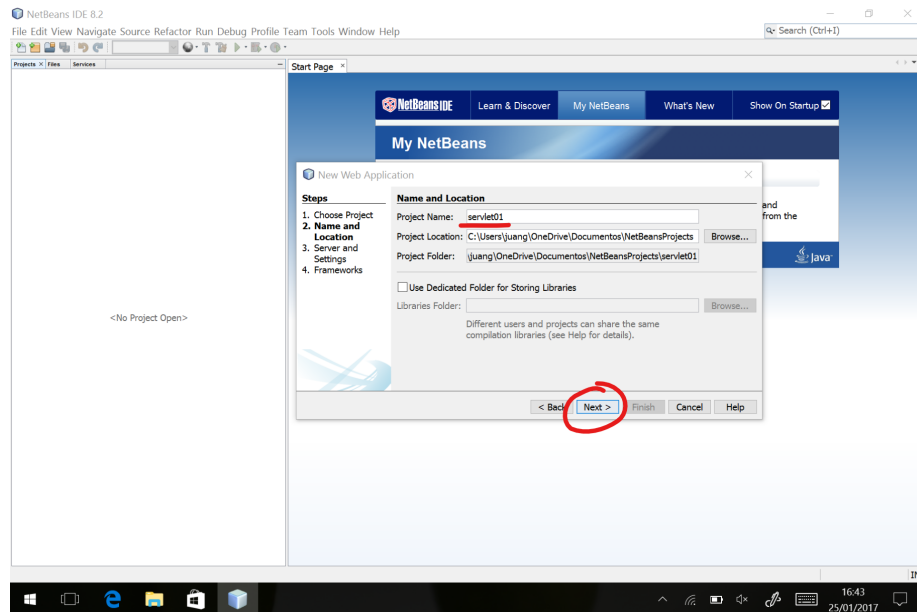


Creando el proyecto

Seleccionamos el menú **File-> New project.**



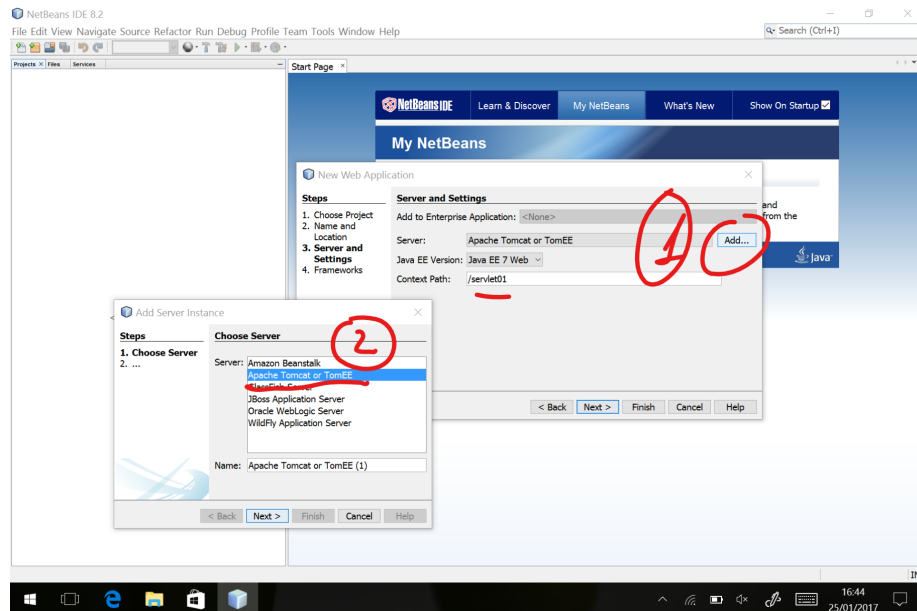
Ahora le damos un nombre al proyecto y seleccionamos dónde queremos guardarlo en nuestro disco duro. Pulsamos **Next**.



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:

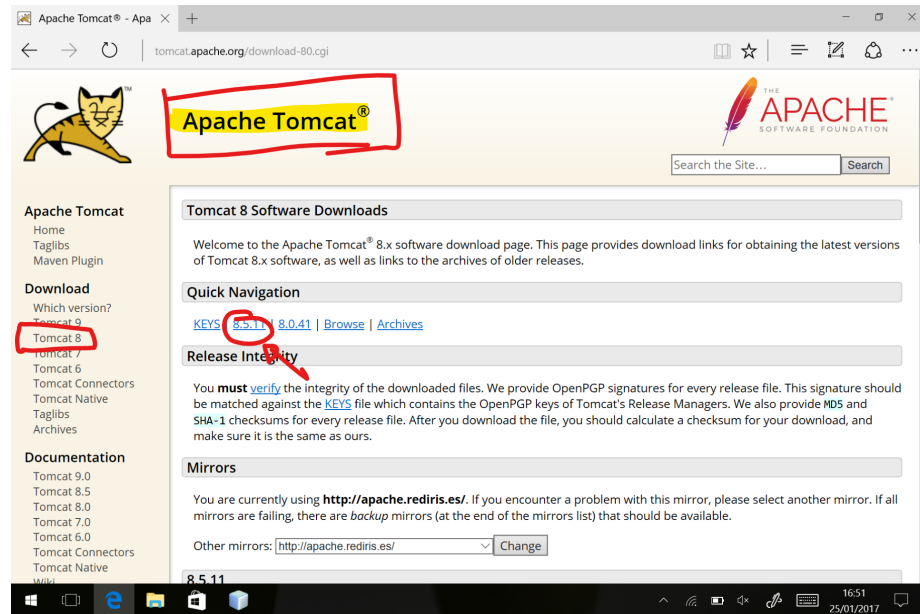
Apache Tomcat



Descarga e instalación de Tomcat.

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

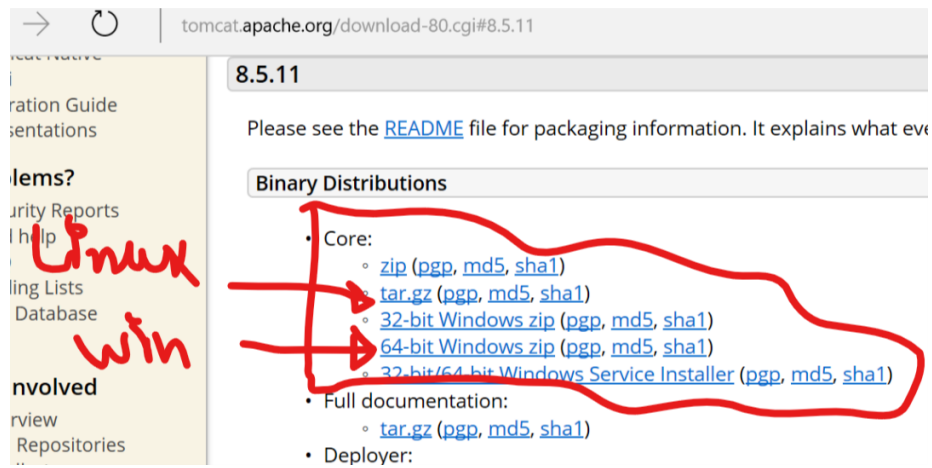
En nuestro caso nos decantamos por la versión 8.5.XX.



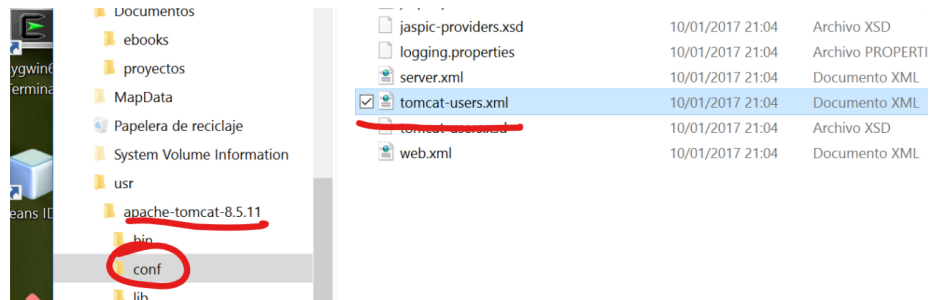
Seleccionamos el “tar.gz” si estamos en un sistema *NIX o MAC y el ZIP en Windows.

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

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

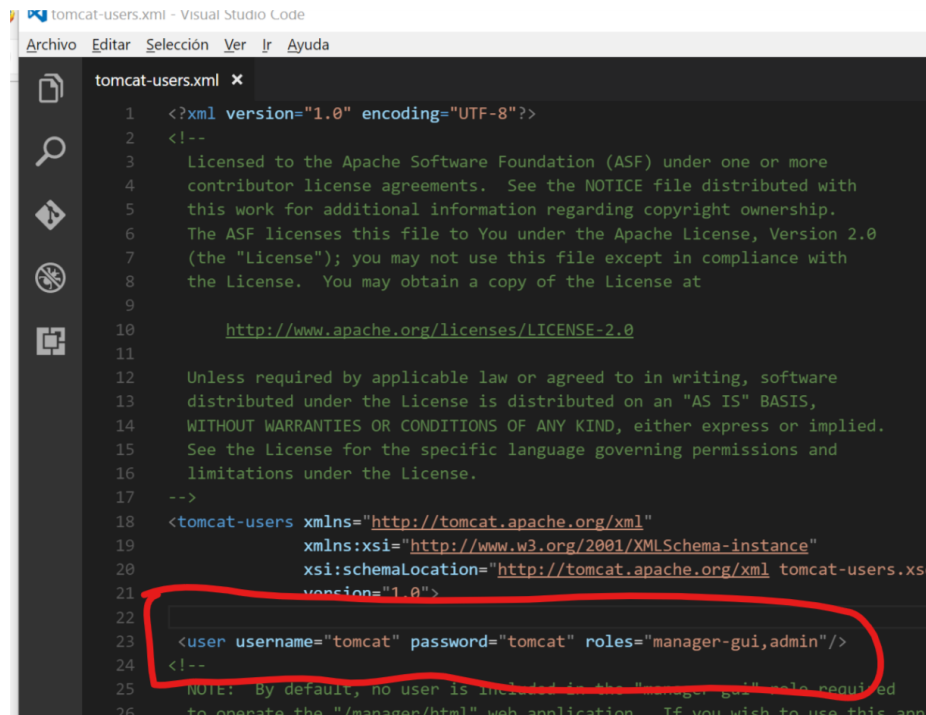


Una vez descomprimido, hemos de configurar el acceso de administrador para poder subir aplicaciones (ficheros WAR) al servidor. Para ello editamos el fichero **tomcat-users.xml** que está en la carpeta **conf**.



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

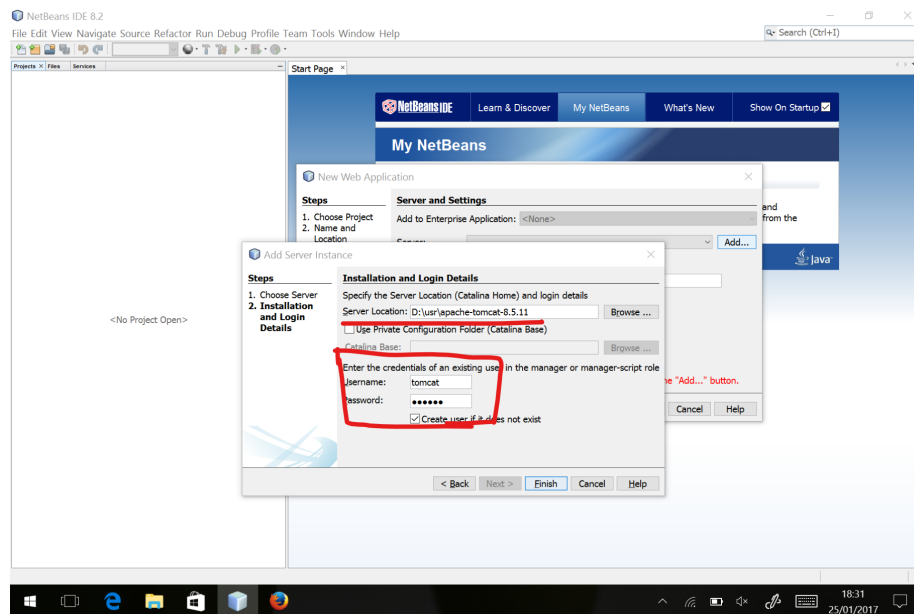
```
1 <user username="tomcat" password="tomcat" roles="admin,
    manager-gui"/>
```



```
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.xsd"
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
```

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
6  <!-- Servlet para la gestión de usuarios -->

```

```

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

```

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     @Override
28     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     // BORRAR
42     @Override
43     protected void doDelete(HttpServletRequest req, // parámetros de
44         la petición
45         HttpServletResponse resp) // respuesta que genero
46         throws ServletException, IOException {
47     }
48
49     @Override
50     public void destroy() {
51     }
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
        password){
21        this.id = id;
22        this.username = username;
23        this.password = password;
24        this.email = email;
25    }
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.LocalDateTime;
4
5 public class Horario {
6     Instalacion instalacion;
7     LocalDateTime inicio;
8     LocalDateTime fin;
9     Long id;
10
11     public Horario() {
12     }
13
14     public Horario(Long id, Instalacion instalacion, LocalDateTime
        inicio, LocalDateTime fin) {
15         this.instalacion = instalacion;
16         this.inicio = inicio;
17         this.fin = fin;
18         this.id = id;
19     }
20
21     public Instalacion getInstalacion() {
22         return this.instalacion;
23     }
24
25     public void setInstalacion(Instalacion instalacion) {
26         this.instalacion = instalacion;
27     }
28
29     public LocalDateTime getInicio() {
30         return this.inicio;
31     }
32
33     public void setInicio(LocalDateTime 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;

```

```

24         this.horario = horario;
25         this.fecha = fecha;
26     }
27
28     public Long getId() {
29         return this.id;
30     }
31
32     public void setId(Long id) {
33         this.id = id;
34     }
35
36     public Usuario getUsuario() {
37         return this.usuario;
38     }
39
40     public void setUsuario(Usuario usuario) {
41         this.usuario = usuario;
42     }
43
44     public Horario getHorario() {
45         return this.horario;
46     }
47
48     public void setHorario(Horario horario) {
49         this.horario = horario;
50     }
51
52     public Date getFecha() {
53         return this.fecha;
54     }
55
56     public void setFecha(Date fecha) {
57         this.fecha = fecha;
58     }
59
60     public Reserva id(Long id) {
61         this.id = id;
62         return this;
63     }
64
65     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         }
87         Reserva reserva = (Reserva) o;
88         return usuario.equals(reserva.usuario) &&
89             horario.equals(reserva.horario) &&
90             fecha.compareTo(reserva.fecha)!=0;
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     }
16     public Connection getConnection() {
17         return conn;
18     }
19 }
```

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
2 import java.sql.Connection;
3 import java.sql.SQLException;
4 import javax.naming.Context;
5 import javax.naming.InitialContext;
6 import javax.sql.DataSource;
7
8 /**
9  *
10  * @author juangu
11  */
12 public class Conexion {
13
14     Connection conn;
15     Context ctx;
16     DataSource ds;
```

```

17
18     public Conexion() {
19         // Vía DataSource con Contexto inyectado
20         try {
21             if (ctx == null)
22                 ctx = new InitialContext();
23             if (ds == null)
24                 ds = (DataSource) ((Context) ctx.lookup(
25                     "java:comp/env")).lookup("jdbc/gestionReservas");
26             conn = ds.getConnection();
27         } catch (Exception ex) {
28             System.out.println("## Conexion ERROR ## " +
29                 ex.getLocalizedMessage());
30             ctx = null;
31             ds = null;
32             conn = null;
33         }
34     }
35
36     public Connection getConnection() {
37         return conn;
38     }

```

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, Glashfish...) 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

Abrir la conexión desde el método init()

```

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      }

```

Cerrar la conexión desde el método destroy()

```

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     public Connection getConnection() {
18         return conn;
19     }
20 }

```

CRUD básico

LEER uno (findOne)

```
1 String jsonObject="{ }";
2 Connection conexion;
3 PreparedStatement pstmt;
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    pstmt = conexion.prepareStatement(sql);
14    pstmt.setInt(1, Integer.parseInt(id));
15    ResultSet rs = pstmt.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 }
31 // 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"
4     pageEncoding="UTF-8" %>
5
6 <sql:query var="userList" dataSource="jdbc/gestionReservas">
7     select username, email, password from usuario;
8 </sql:query>
```

```

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 pstmt;
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)
10             VALUES(?,?,?)";
11         pstmt = conexion.prepareStatement(sql);
12         pstmt.setString(1, username);
13         pstmt.setString(2, password);
14         pstmt.setString(3, email);
15         if (pstmt.executeUpdate() >0) {
16             // "Usuario insertado"
17         } else {
18             // "No se ha podido insertar"
19         }
20         conexion.close();
21     } catch (Exception ex) {
22         // "Imposible conectar a la BBDD"
23     }

```

Actualizar

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

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",
        "example");
9      String sql = "DELETE FROM usuario WHERE id=?";
10     pstm = conexion.prepareStatement(sql);
11     pstm.setInt(1, Integer.parseInt(id));
12     if ( pstm.executeUpdate()==0 ) {
13         jsonObject="{ "+
14             "'id':'"+id+"'}";
```

```

15     }
16     }catch(Exception ex){
17         // "No se pudo eliminar"
18     }

```

CRUD (patrón DAO)

Ya tenemos las clases base que contiene los objetos que vamos a almacenar/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> findByNombre(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.err.println("\nID: " + id + "\n");
11            ResultSet rs = pstmt.executeQuery();
12            rs.next();
13            instala = new Instalacion(rs.getInt("id"),
14                                     rs.getString("nombre"));
14            conn.close();

```

```

15     } catch (SQLException ex) {
16         instala = null;
17     }
18     return instala;
19 }
20
21 @Override
22 public List<Instalacion> findAll() {
23     Instalacion instala;
24     List<Instalacion> li_ins = new ArrayList<Instalacion>();
25     try {
26         Conexion conexion = new Conexion();
27         Connection conn = conexion.getConnection();
28
29         String sql = "SELECT * FROM instalacion";
30
31         PreparedStatement pstmt = conn.prepareStatement(sql);
32
33         ResultSet rs = pstmt.executeQuery();
34         // 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;
47 }

```

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
6   * @param new_al El objeto que contiene los datos nuevos
7   *               delinstalaummo
8   * @return true si se lleva a cabo correctamente <br>
9   *         false si no se actualiza nada (error de conexión, no
10   *         estaba
11   *         elinstalaummo en la BBDD...) <br>
12   */
13  @Override
14  public boolean update(Instalacion old_al, Instalacion new_al) {
15
16      return update(old_al.getId(), new_al);
17  }
18
19  /**
20   * Este método actualiza una instalación en la BBDD
21   *
22   * @param old_id El id antiguo delinstalaummo
23   * @param new_al El objeto que contieneinstalainstalaummo
24   *               actualizado
25   * @return true si se lleva a cabo correctamente <br>
26   *         false si no se actualiza nada (error de conexión, no
27   *         estaba
28   *         elinstalaummo en la BBDD...) <br>
29   */
30  @Override
31  public boolean update(Integer old_id, Instalacion new_al) {
32      boolean exito = true;
33      try {
34          Conexion conexion = new Conexion();
35          Connection conn = conexion.getConnection();

```



```

31         String sql = "UPDATE instalacion SET id=?, nombre=?
                       WHERE id=?";
32         PreparedStatement pstmt = conn.prepareStatement(sql);
33         pstmt.setInt(3, old_id);
34         pstmt.setInt(1, new_al.getId());
35         pstmt.setString(2, new_al.getName());
36         if (pstmt.executeUpdate() == 0) {
37             exito = false;
38         }
39         conn.close();
40     } catch (SQLException ex) {
41         exito = false;
42     }
43     return exito;
44 }

```

Borrar (InstalacionDaoImpl.java)

```

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

```

```

27         conn.close();
28     } catch (SQLException ex) {
29         exito = false;
30     }
31     return exito;
32 }

```

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 inyectar 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 *logueados*.

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"
                placeholder="Contraseña" />
            <button type="submit">Enviar</button>

```

```

8         </form>
9     </div>
10 </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 pstmt = conexion.prepareStatement(sql);
22              pstmt.setString(1, user);
23              pstmt.setString(2, pwd);
24              ResultSet rs = pstmt.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

```

```

38         wrong."+"</font>");
39         rd.include(request, response);
40     }
41     conexion.close();
42 } catch (SQLException e) {
43     // response.sendRedirect("login.jsp");
44     response.getWriter().print(e.getLocalizedMessage());
45 }
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
2 public class AuthenticationFilter implements javax.servlet.Filter {
3     public static final String AUTHENTICATION_HEADER =
4         "Authorization";
5
6     @Override
7     public void doFilter(ServletRequest request, ServletResponse
8         response,
9         FilterChain filter) throws IOException, ServletException
10    {
11        if (request instanceof HttpServletRequest) {
12            HttpServletRequest httpRequest =
13                (HttpServletRequest) request;

```

```

10
11     Cookie loginCookie = null;
12     Cookie[] cookies = httpRequest.getCookies();
13     if (cookies != null) {
14         for (Cookie cookie : cookies) {
15             if (cookie.getName().equals("ges_res.user")) {
16                 loginCookie = cookie;
17                 break;
18             }
19         }
20     }
21     if (loginCookie != null) {
22         filter.doFilter(request, response);
23     } else {
24         //
25         ((HttpServletResponse)response).sendRedirect("login.jsp");
26         if (response instanceof HttpServletResponse) {
27             HttpServletResponse httpResponse =
28                 (HttpServletResponse) response;
29             httpResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
30             httpResponse.sendRedirect("/login.jsp");
31         }
32     }
33 }
34
35 @Override
36 public void destroy() {
37 }
38
39 @Override
40 public void init(FilterConfig arg0) throws ServletException {
41 }
42 }

```

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;

```

```

7      Cookie[] cookies = request.getCookies();
8      if (cookies != null) {
9          for (Cookie cookie : cookies) {
10             if (cookie.getName().equals("ges_res.user")) {
11                 loginCookie = cookie;
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/instalacion
Ver el detalle de una instalación	GET	/api/instalacion/{id}
Crear una nueva instalación	POST	/api/instalacion
Borrar una instalación	DELETE	/api/instalacion/{id}
Buscar una instalación por nombre	GET	/api/instalacion/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 **automáticamente** 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: 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   */
7
8  package com.iesvdc.acceso.simplecrud.model;
9
10 import java.sql.Connection;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.sql.SQLException;
14 import java.util.ArrayList;
15 import java.util.List;
16
17 import javax.naming.Context;
18 import javax.naming.InitialContext;
19 import javax.sql.DataSource;
20
21 /**

```

```

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 findById(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.err.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 Instalacions en la BBDD por nombre:
103  * @param nombre
104  * El nombre a buscar
105  * @return
106  * Devuelve:
107  * null: si hay instalagún error (no se puede conectar a la
108  * BBDD...). <br>
109  * ArrayList vacío (length == 0): si no hay nadie con ese
110  * nombre. <br>

```

```

109      * ArrayList con Instalaciones: si hay instalauomos con ese
110      nombre.<br>
111      */
112      public List<Instalacion> findByNombre(String nombre){
113          Instalacion instala;
114          List<Instalacion> li_ins = new ArrayList();
115          try {
116              // conectamos a la BBDD
117              Conexion conexion = new Conexion();
118              Connection conn = conexion.getConnection();
119              // esta es la cadena SQL de conslulta
120              String sql = "SELECT * FROM instalacion WHERE nombre=?";
121              // usamos este objeto porque es más seguro
122              PreparedStatement pstmt = conn.prepareStatement(sql);
123              pstmt.setString(1, nombre);
124              // ejecutar la consulta contra la base de datos y
125              // devuelve el resultado en el ResultSet (parecido a
126              // un Array con iterador
127              ResultSet rs = pstmt.executeQuery();
128              // recorro el resultset mientras tengo datos
129              while (rs.next()){
130                  instala = new Instalacion(
131                      rs.getInt("id"),
132                      rs.getString("nombre"));
133                  li_ins.add(instala);
134              }
135              // cerramos la conexión
136              conn.close();
137          } catch (SQLException ex) {
138              System.out.println("ERROR"+ ex.getMessage());
139              li_ins = null;
140          }
141          return li_ins;
142      }
143
144
145
146      /**
147      * Este método actualiza un instalauomo en la BBDD
148      * @param old_al
149      * El objeto que contiene los datos antiguos del instalauomo
150      * @param new_al
151      * El objeto que contiene los datos nuevos del instalauomo
152      * @return
153      * true si se lleva a cabo correctamente <br>

```

```

154      * false si no se actualiza nada (error de conexión, no
155      * estaba elinstallaumno 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 delinstallaumno
166       * @param new_al
167       * El objeto que contieneinstallainstalaumno 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 elinstallaumno 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 instalainstalaumno a borrar
199       * @return

```

```

200      * true si borra uninstalaummo <br>
201      * false si elinstalaummo no existe o no se puede conectar a la
        BBDD <br>
202      */
203      public boolean delete(Instalacion instala){
204          return delete(instala.getId());
205      }
206
207      public boolean delete(Integer id_al){
208          boolean exito=true;
209          try {
210              Conexion conexion = new Conexion();
211              Connection conn = conexion.getConnection();
212              String sql = "DELETE FROM instalacion WHERE id=?";
213              PreparedStatement pstmt = conn.prepareStatement(sql);
214              pstmt.setInt(1, id_al);
215              if (pstmt.executeUpdate()==0) {
216                  exito = false;
217              }
218              conn.close();
219          } catch (SQLException ex) {
220              exito = false;
221          }
222          return exito;
223      }
224
225  }

```

El servicio web (WS)

Primero creamos el recurso que usará Jersey:

```

1
2 package com.iesvdc.acceso.simplecrud.controller.service;
3
4 import com.iesvdc.acceso.simplecrud.model.*;
5
6 import java.util.ArrayList;
7 import java.util.List;
8 import java.util.logging.Logger;
9 import javax.ws.rs.Consumes;
10 import javax.ws.rs.GET;
11 import javax.ws.rs.POST;
12 import javax.ws.rs.PUT;
13 import javax.ws.rs.DELETE;
14 import javax.ws.rs.Path;

```

```

15 import javax.ws.rs.PathParam;
16 import javax.ws.rs.Produces;
17 import javax.ws.rs.core.MediaType;
18 import javax.ws.rs.core.Response;
19
20 /**
21  *
22  * @author Juangu <jgutierrez at iesvirgencelcarmen.com>
23  */
24 @Path("/")
25 public class InstalacionResource {
26
27     @GET
28     @Path("instalacion")
29     @Produces({MediaType.APPLICATION_JSON,
30         MediaType.APPLICATION_XML})
31     public List<Instalacion> getInstalaciones() {
32         InstalacionDAO al_dao = new InstalacionDAO();
33         List<Instalacion> list_al;
34         try {
35             list_al = al_dao.findAll();
36         } catch (Exception ex) {
37             list_al = new ArrayList<>();
38             Logger.getLogger(ex.getLocalizedMessage());
39         }
40         return list_al;
41     }
42
43     @GET
44     @Path("instalacion/{id}")
45     @Produces({MediaType.APPLICATION_JSON,
46         MediaType.APPLICATION_XML})
47     public Instalacion getInstalacionById(@PathParam("id") String
48         id) {
49         InstalacionDAO al_dao = new InstalacionDAO();
50         Instalacion al;
51         try {
52             al = al_dao.findById(Integer.parseInt(id));
53         } catch (Exception ex) {
54             al = new Instalacion(-1, "Error");
55             Logger.getLogger(ex.getLocalizedMessage());
56         }
57         return al;
58     }
59 }

```

```

58  @GET
59  @Path("instalacion/nombre/{nombre}")
60  @Produces({MediaType.APPLICATION_JSON,
61             MediaType.APPLICATION_XML})
62  public List<Instalacion>
63      getInstalacionByNombre(@PathParam("nombre") String nombre) {
64      InstalacionDAO al_dao = new InstalacionDAO();
65      List<Instalacion> list_al;
66      try {
67          list_al = al_dao.findByNombre(nombre);
68      } catch (Exception ex) {
69          list_al = new ArrayList<>();
70          Logger.getLogger(ex.getLocalizedMessage());
71      }
72      return list_al;
73  }
74
75  @PUT
76  @Path("instalacion/{id}")
77  @Consumes({MediaType.APPLICATION_JSON,
78             MediaType.APPLICATION_XML})
79  public void updateInstalacion(@PathParam("id") Integer id,
80                               Instalacion al) {
81      InstalacionDAO al_dao = new InstalacionDAO();
82      try {
83          al_dao.update(id, al);
84      } catch (Exception ex) {
85          Logger.getLogger(ex.getLocalizedMessage());
86      }
87  }
88
89  @POST
90  @Consumes({MediaType.APPLICATION_JSON,
91             MediaType.APPLICATION_XML})
92  @Produces({MediaType.APPLICATION_JSON,
93             MediaType.APPLICATION_XML})
94  @Path("alumno")
95  public Response createInstalacion(Instalacion al) {
96      InstalacionDAO al_dao = new InstalacionDAO();
97      try {
98          al_dao.create(al);
99      } catch (Exception ex) {
100          Logger.getLogger(ex.getLocalizedMessage());
101          return Response.status(400).entity(al).build();

```



```

98     }
99     return Response.status(200).entity(al).build();
100 }
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 }
112 }

```

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".

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

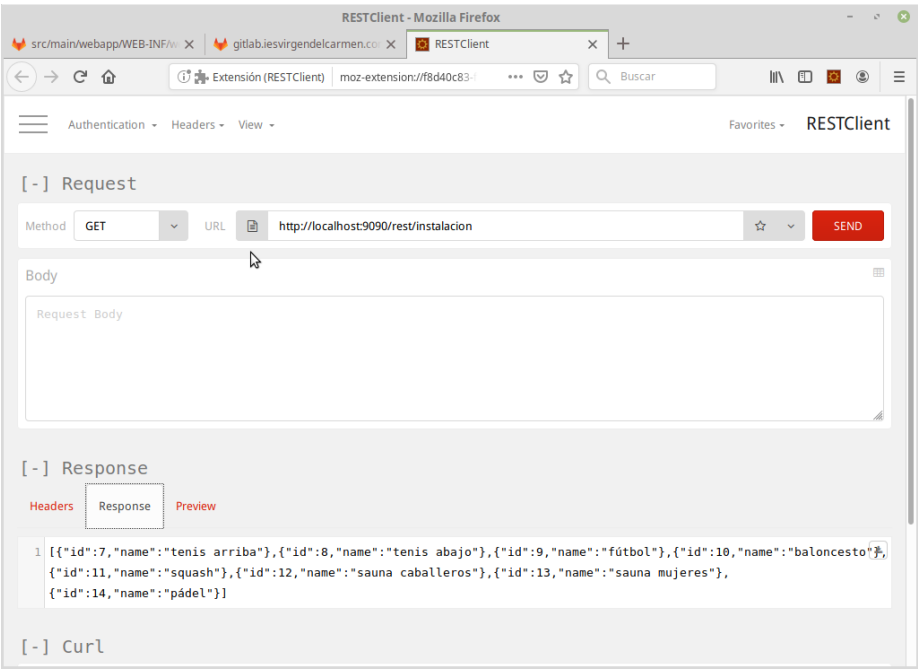


Figure 2: Extensión RESTClient de firefox

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/instalacion
Ver el detalle de una instalación	GET	/api/instalacion/{id}
Crear una nueva instalación	POST	/api/instalacion
Borrar una instalación	DELETE	/api/instalacion/{id}
Buscar una instalación por nombre	GET	/api/instalacion/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.

58

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;
```

```

20     response.setHeader("Access-Control-Allow-Methods", "POST,
        GET, PUT, DELETE");
21     response.setHeader("Access-Control-Max-Age", "3600");
22     response.setHeader("Access-Control-Allow-Headers",
        "Origin, x-requested-with, Content-Type, Accept");
23     chain.doFilter(req, res);
24 }
25
26
27 public void init(FilterConfig filterConfig) {}
28
29 public void destroy() {}
30
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 reservapp com.iesvdc.dam.acceso ReservAPP
3 $ cd reservapp
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 class="panel" id="panel_inst_delete">

```

```

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

```

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

```

```

2 $.controller.active_panel = "#panel_inicio";
3
4 $.controller.activate = function (panel_name) {
5     $($.controller.active_panel).hide();
6     $(panel_name).show();
7     $.controller.active_panel = panel_name;
8 };
9
10 $.controller.init = function (panel_inicial) {
11     $('[id^="menu_"]').each(function () {
12         var $this = $(this);
13         var menu_id = $this.attr('id');
14         var panel_id = menu_id.replace('menu_', 'panel_');
15
16         $("#" + menu_id).click(function () {
17             $.controller.activate("#" + panel_id);
18         });
19     });

```

Verbos HTTP comunicación cliente/servidor

Para comunicarnos con el servidor desde el cliente, usaremos los verbos HTTP:

- GET: leer
- POST: crear
- PUT: actualizar
- DELETE: borrar
- OPTIONS: consultar disponibilidad (verbos disponibles)

A tal efecto, y por simplificar, hemos implementado las llamadas en nuestro controlador y las enumeramos a continuación:

GET

```

1 /**
2  * Función para hacer el GET al servicio REST
3  * @param {type} target la URL donde está el servicio REST
4  * @param {type} fn_exito función a llamar cuando tenga éxito
5  */
6 $.controller.doGet = function (target, fn_exito) {
7     $.get({
8         url: target,
9         type: 'GET',
10        dataType: 'json',
11        contentType: 'application/json;charset=UTF-8',
12        crossDomain: true,
13        success: fn_exito,

```

```

14     beforeSend: function (xhr) {
15         $.controller.authorize(xhr);
16     },
17     error: function (xhr, status) {
18         $.controller.errorManager(xhr.status);
19     }
20 };

```

POST

```

1 /**
2  * Función para hacer el POST al servicio REST
3  * @param {type} target la URL donde está el servicio REST
4  * @param {type} datos los datos a subir
5  * @param {type} fn_exito función a llamar cuando tenga éxito
6  */
7 $.controller.doPost = function(target, datos, fn_exito) {
8     $.ajax({
9         url: target ,
10        method: 'POST',
11        dataType: 'json',
12        contentType: "application/json;charset=UTF-8",
13        data: JSON.stringify(datos),
14        beforeSend: function (xhr) {
15            $.controller.authorize(xhr);
16        },
17        error: function (xhr, status) {
18            $.controller.errorManager(xhr.status);
19        }
20    }).done(fn_exito);
21 };

```

PUT

```

1 /**
2  *
3  * @param {type} target
4  * @param {type} id
5  * @param {type} datos
6  * @param {type} fn_exito
7  */
8 $.controller.doPut = function(target, id, datos, fn_exito) {
9     $.ajax({
10        url: target+'/'+id,
11        method: 'PUT',
12        dataType: 'json',

```

```

13     contentType: "application/json;charset=UTF-8",
14     data: JSON.stringify(datos),
15     beforeSend: function (xhr) {
16         $.controller.authorize(xhr);
17     },
18     success: fn_exito,
19     error: function (xhr, status) {
20         $.controller.errorManager(xhr.status);
21     }
22 });
23 };

```

DELETE

```

1 /**
2  *
3  * @param {type} target
4  * @param {type} id
5  * @param {type} fn_exito
6  */
7 $.controller.doDelete = function(target, id, fn_exito) {
8     $.ajax({
9         url: target + '/' + id,
10        method: 'DELETE',
11        dataType: 'json',
12        contentType: "application/json;charset=UTF-8",
13        beforeSend: function (xhr) {
14            $.controller.authorize(xhr);
15        },
16        success: fn_exito,
17        error: function (xhr, status) {
18            $.controller.errorManager(xhr.status);
19        }
20    });
21 };

```

Autenticación

Con este modelo, si queremos, por ejemplo, hacer autenticación básica, o cualquier otra, simplemente la implementamos en este método:

```

1 /**
2  * Función para gestionar la autorización contra el servidor.
3  * Puedes cambiarla para hacerlo con sesiones, sessionStorage,
4  * token, OAuth...
5  * @param {type} xhr las cabeceras

```



```

6  */
7  $.controller.authorize = function(xhr) {
8      xhr.setRequestHeader ("Authorization", "Basic " +
9          btoa($.controller.username+":"+$.controller.password));
10 };

```

CRUD Instalación (front-end)

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/instalacion.js`), posiblemente tengamos que añadir algún panel más al fichero `index.html`:

```

1  $.instalacion.init = function() {
2
3      // Lógica de la aplicación: Instalacion.findAll()
4      $("#menu_inst_read").click(=>{
5          console.log("listando instalaciones... ");
6          $.instalacion.findAll();
7      });
8
9      // Lógica de la aplicación: Instalacion.delete()
10     $("#menu_inst_delete").click(=>{
11         $.instalacion.delete();
12     });
13
14     // Lógica de la aplicación: Instalacion.update()
15     $("#menu_inst_update").click(=>{
16         $.instalacion.update();
17     });
18
19     // Lógica de la aplicación: Instalacion.create()
20     $("#btn_inst_create").click(=>{
21         //$("#nombre_inst_create").val("");
22         $.instalacion.create();
23     });
24
25     // botón formulario con datos actualizados
26     $("#btn_inst_update").click(=>{
27         //console.log("btn_inst_update-onClick:TODO:: falta llamar
28             al doGet "+
29             // "para poblar el formulario y luego hacer el doPut");
30         $.controller.activate("#panel_inst_update_form");
31         $("#id_inst_update_form").val($("#select_inst_update").val());
32         $("#nombre_inst_update_form").val($("#select_inst_update
33             option:selected").text());

```

```

32     });
33
34     // Lógica de la aplicación: Instalacion.updateForm()
35     $("#btn_inst_update_form").click(()=>{
36         $.instalacion.update($("#id_inst_update_form").val());
37     });
38
39     $("#btn_inst_delete").click(()=>{
40         // console.log("btn_inst_delete-onClick::TODO:: falta llamar
           al doDelete");
41
42         $.controller.modal(
43             "Borrar instalación",
44             "Si elimina una instalación se borrarán también "+
45             "sus horarios y reservas, ¿está realmente seguro?",
46             () => {
47                 $.instalacion.delete($("#select_inst_delete").val());
48             });
49     });
50 }

```

Ya está hecho **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.

Instalación findAll():

```

1     $.instalacion.findAll = () => {
2         $.controller.doGet($.controller.url+"instalacion",
           function(datos){
3             console.log("listando instalaciones: "+datos);
4             $("#panel_inst_list").empty();
5             let tabla=$("#<table class='table' />");
6             tabla.append("<thead><tr><th>#</th><th>nombre</th><th>Act.</th><th>Borr.</th></tr></thead>");
7             datos.forEach(instalacion => {
8                 let fila=$("#<tr>");
9                 fila.append("<td>"+instalacion.id+"</td>");
10                fila.append("<td>"+instalacion.name+"</td>");
11                fila.append("<td>"+<span class='btn btn-success'
                   onclick='$.instalacion.updateForm("+
12                instalacion.id+"')>ACTUALIZAR</span> "+</td>");
13                fila.append("<td><span class='btn btn-danger'
                   onclick='$.instalacion.delete("+
14                instalacion.id+"')>BORRAR</span> </td>");
15                tabla.append(fila);
16            });

```

```

17     $("#panel_inst_list").append(tabla);
18   });
19 }

```

Instalación create():

```

1  $.instalacion.create = () => {
2    let datos = {
3      id: 0,
4      name: $("#nombre_inst_create").val()
5    }
6    $.controller.doPost($.controller.url+"instalacion",datos, () => {
7      // $.controller.activate_inicio();
8      $.instalacion.findAll();
9      $.controller.activate("#panel_inst_read");
10     $.controller.alert("Instalación:", " Añadida
        correctamente.");
11     $("#nombre_inst_create").val("");
12   });
13 }

```

Instalación update():

```

1  $.instalacion.update = (id) => {
2    if (typeof id === "undefined") {
3      $.controller.doGet($.controller.url+"instalacion",(datos)=>{
4        $("#select_inst_update").empty();
5        console.log("preparando delete");
6        datos.forEach(instalacion => {
7          $("#select_inst_update").append('<option value="'+
8            instalacion.id+'">'+
9            instalacion.name+
10           "</option>");
11        });
12      });
13    } else {
14      let datos = {
15        id: $("#id_inst_update_form").val(),
16        name: $("#nombre_inst_update_form").val()
17      }
18      $.controller.doPut($.controller.url+"instalacion", datos.id,
19        datos, ()=>{
20        $.instalacion.findAll();
21        $.controller.activate("#panel_inst_read");
22        $.controller.alert("Instalación:", " Actualizada
            correctamente.");
23      });
24    }
25  }

```

```

23     }
24 };
25
26 $.instalacion.updateForm = (id) => {
27     $.controller.doGet($.controller.url+"instalacion/"+id, (datos)
28         => {
29         $("#id_inst_update_form").val(datos.id);
30         $("#nombre_inst_update_form").val(datos.name);
31         $.controller.activate("#panel_inst_update_form");
32     });
33 };

```

Instalación delete():

```

1 $.instalacion.delete = (id) => {
2     if (typeof id === "undefined") {
3         $.controller.doGet($.controller.url+"instalacion", (datos)=>{
4             $("#select_inst_delete").empty();
5             console.log("preparando delete");
6             datos.forEach(instalacion => {
7                 $("#select_inst_delete").append('<option value="'+
8                     instalacion.id+'">'+
9                     instalacion.name+
10                     "</option>");
11             });
12         });
13     } else {
14         $.controller.doDelete($.controller.url+"instalacion",
15             id, () => {
16                 $.instalacion.findAll();
17                 $.controller.activate("#panel_inst_read");
18             });
19     }
20 };

```

Localtunnel

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
2   pagar a final de mes...
3
4 DROP TRIGGER IF EXISTS reserva_semanal;
5
6 DELIMITER $$
7 CREATE TRIGGER `reserva_semanal`
8 BEFORE INSERT ON `reserva` FOR EACH ROW
9 BEGIN
10     IF ( NEW.`fecha` < CURDATE())
11     THEN
12         SIGNAL sqlstate '45002'
13         SET message_text = 'No se permite reservar en una fecha
14                             anterior a la actual.';
15     ELSEIF ( NEW.`fecha` > DATE_ADD(CURDATE(), INTERVAL 14 DAY) )
16     THEN
17         SIGNAL sqlstate '45003'
18         SET message_text = 'No se permite reservar con más de dos
19                             semanas de antelación.';
20     END IF;
21 END;
22 $$

```

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]
```