

PRÁCTICA 2 - PTI

Kais Harim, Omair Iqbal

23.02.2017

INTRODUCCIÓN	2
SERVLET	2
TOMCAT	2
RECURSOS	3
PROCEDIMIENTO	3
EJERCICIO 1	3
EJERCICIO 2	3
CONCLUSIONES	3

INTRODUCCIÓN

El objetivo de esta sesión es trabajar con servlet de Java y conocer el funcionamiento de servidores tipo Tomcat.

SERVLET

El servlet es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor. Aunque los servlets pueden responder a cualquier tipo de solicitudes, éstos son utilizados comúnmente para extender las aplicaciones alojadas por servidores web, de tal manera que pueden ser vistos como applets de Java que se ejecutan en servidores en vez de navegadores web. Este tipo de servlets son la contraparte Java de otras tecnologías de contenido dinámico Web, como PHP y ASP.NET.

La palabra servlet deriva de otra anterior, applet, que se refiere a pequeños programas que se ejecutan en el contexto de un navegador web.

El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

TOMCAT

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets, descrito anteriormente, y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por Sun Microsystems).

Tomcat es un contenedor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

RECURSOS

1. Git de la asignatura
2. Servidor Tomcat
3. Herramientas para editar documentos java

PROCEDIMIENTO

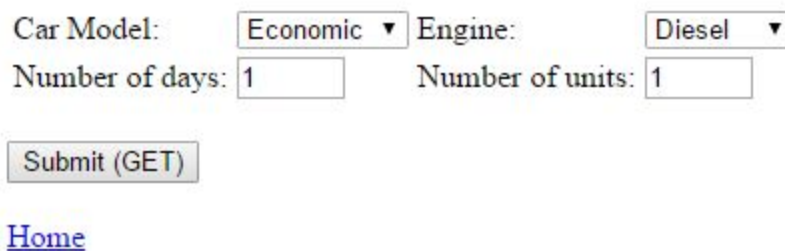
En primer lugar hemos instalado Java y el servidor Tomcat en nuestra máquina siguiendo la guía de la práctica.

Una vez instalado el servidor y puesto todo a punto hemos comenzado a programar la web que pedía el enunciado.

EJERCICIO 1

Para la primera parte del ejercicio, hemos usado la plantilla proporcionada. En primer lugar, recopilamos los parámetros de la request en el método doGet. Una vez obtenidos los valores de la petición los incorporamos a un objeto JSON, para poder tratar con los datos desde el servidor y almacenarlos en la base de datos.

New rental



The screenshot shows a web form titled "New rental". It contains four input fields: "Car Model:" with a dropdown menu showing "Economic", "Engine:" with a dropdown menu showing "Diesel", "Number of days:" with a text input containing "1", and "Number of units:" with a text input containing "1". Below these fields is a button labeled "Submit (GET)". At the bottom left of the form is a blue underlined link labeled "Home".

Figura 1: Página web para hacer una reserva

Al darle al botón “Submit (GET)” se ejecutará la función doGet y almacenará los campos rellenos en la web en las variables que vemos a continuación

```

public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    String ModelVehicle = req.getParameter("model_vehicle");
    String SubModel = req.getParameter("sub_model_vehicle");
    String numberDays = req.getParameter("dies_lloguer");
    String numberUnits = req.getParameter("num_vehicles");

```

Figura 2: método doGet para obtener los parámetros

En función de los parámetros escogidos, vamos actualizando la variable finalPrice para que contenga el precio final del alquiler.

```

    int finalPrice = 0;
    if (SubModel.equals("Diesel")) finalPrice += 20;
    else finalPrice += 30;

    finalPrice = (Integer.parseInt(ModelVehicle) + finalPrice)

    * Integer.parseInt(numberDays) * Integer.parseInt(numberUnits);

```

Figura 3: cálculo del finalPrice

Ahora que ya tenemos el precio final y los valores de la reserva, el siguiente paso es almacenar los datos en un JSON para finalmente guardarlos en la base de datos, que en este caso es un fichero de texto.

```

JSONObject obj = new JSONObject();
obj.put("modelV", ModelVehicle);
obj.put("subM", SubModel);
obj.put("nDays", numberDays);
obj.put("nUnits", numberUnits);
obj.put("finalP", finalPrice);

myArray.add(obj);

file = new FileWriter("/var/lib/tomcat7/webapps/my_webapp/WEB-INF/classes/mypackage/test.json");
myArray.writeJSONString(file);
file.flush();

out.println("<h1> Your Order </h1>");
out.println("<h2> Final price = " + finalPrice + "</h2>");

```

Figura 4: Creación del JSON con los datos de la reserva

Una vez realizados todos los cálculos, se muestra al usuario los detalles de la reserva.

Your Order
Final price = 74

Figura 5: Se le muestra al usuario el precio final de la reserva

A lo largo del ejercicio hemos ido consultando información sobre cómo tratar con objetos JSON ya que no conocíamos con detalle su funcionamiento. Hemos usado la guía propuesta en la práctica y hemos resuelto los problemas planteados con éxito.

EJERCICIO 2

Para este ejercicio hemos heredado las plantillas existentes del ejercicio de prueba y lo hemos modificado para que cumpla las funcionalidades requeridas. En primer lugar se le pide al usuario que introduzca un usuario y una contraseña válida para poder tener acceso a los datos:

List of rental orders

UserId:

Password:

[Home](#)

Figura 6: Pantalla de autenticación para solicitar la lista de reservas

Una vez comprobado que el usuario y la contraseña son correctos, obtenemos el JSON que tenemos en la base de datos e iteramos sobre todos los objetos para mostrar todas las reservas hechas y mostrarlas por pantalla.

```

try {
    JSONArray a = (JSONArray) parser.parse(new FileReader("/var/lib/tomcat7/webapps/my_webapp/WEB-INF/classes/mypackage/test.json"));

    for (Object o: a)
    {
        JSONObject order = (JSONObject) o;
        String model = "hola";
        String space1 = " ";
        if (order.get("modelV").equals("54")) model = "Economic";
        else if (order.get("modelV").equals("71")) model = "Semi-Luxe";
        else if (order.get("modelV").equals("82")) model = "Luxe";
        else if (order.get("modelV").equals("139")) model = "Limusina";
        out.println("<h2><pre>" + model + space1 + order.get("subM") + space1 + order.get("nDays") +
            space1 + order.get("nDays") + space1 + order.get("nUnits") + space1 + order.get("finalP") + "</pre></h2>");
    }
}

```

Figura 7: Obtención de los parámetros a partir del JSON del fichero de texto

Model Engine Days Units Final price

Economic	Diesel	1	1	1	74
Economic	Diesel	12	12	1	888
Economic	Gasolina	3	3	1	252

Figura 8: Se muestra la lista de reservas al usuario

CONCLUSIONES

Esta práctica nos ha resultado muy útil, ya que nos ha permitido refrescar los skills que teníamos sobre como java a la hora de tratar con servidores, como este Tomcat. Nos ha hecho trabajar en grupo para resolver los problemas que nos hemos encontrado durante la práctica. En definitiva una práctica muy interesante acorde con la materia de la especialidad y muy útil para mejorar nuestras capacidades.