

Práctica 1 PTI: Servlets

Joan Llonch Majó

Luis Jesús Valverde Zavaleta

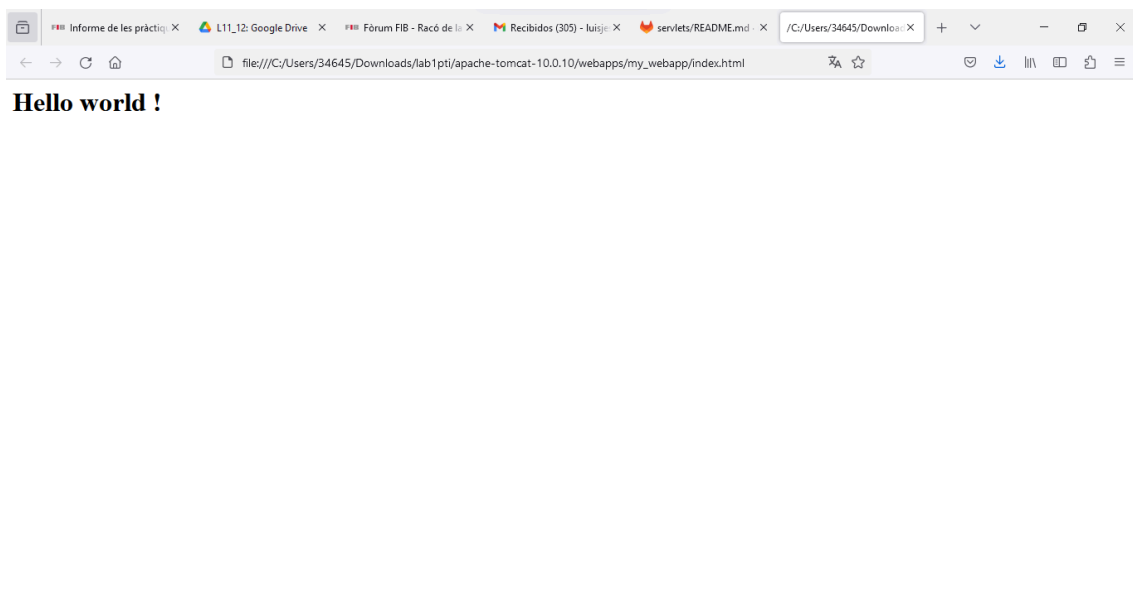
Entorno y configuración de la práctica

El entorno utilizado para la realización de la práctica 1 de PTI ha sido en VirtualBox y con la maquina virtual proporcionada por la UPC que es la Ubunut22v3r1.

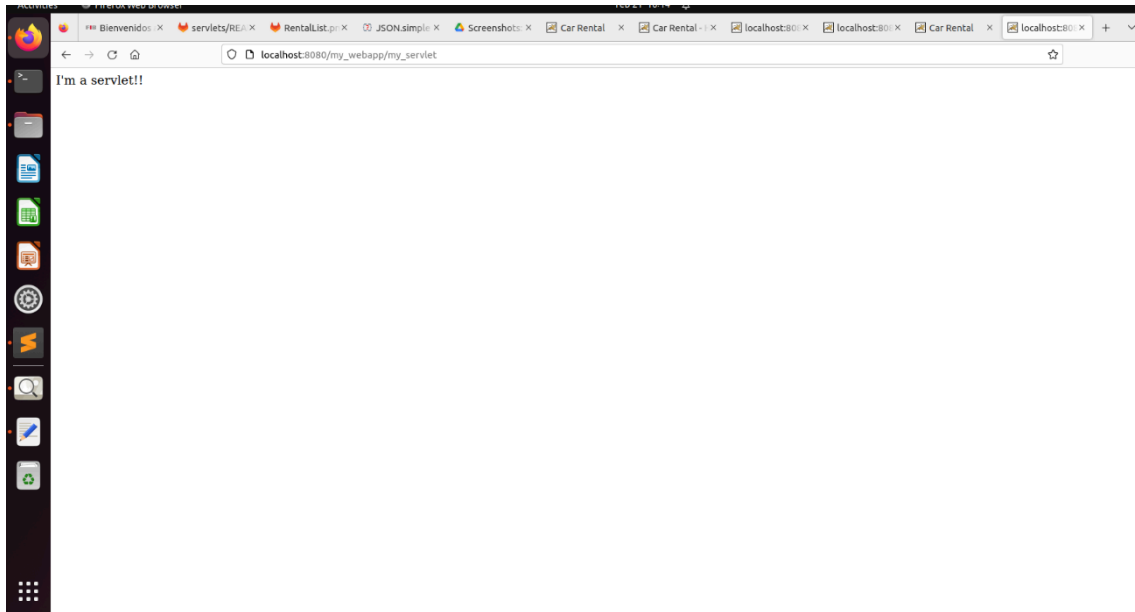
Para la configuración inicial de la práctica hemos seguido los pasos descritos en el enunciado, que principalmente consistían en los siguientes pasos:

1. Desplegar la maquina virtual proporcionada por la UPC, donde esta ya contenía todas las herramientas necesarias instaladas para la realización de la práctica como una versión para desarrollar código en java.
2. Lo segundo que hicimos fue descargar el repositorio de git donde estaban presentes las practicas del curso.
3. Descomprimos con el comando `"tar -xvzf apache-tomcat-10.0.10.tar.gz"` y a partir de allí empezamos a trabajar dentro del directorio descomprimido.
4. Una vez dentro pusimos en marcha el tomcat 10 con el siguiente comando: `"./bin/startup.sh &"`. Y de esta manera ya tenemos un contenedor web con soporte servlets operativo a partir del cual ir desarrollando nuestro proyecto.

Antes de ponernos manos a la obra con la parte principal del enunciado, hicimos unas pruebas html para empezar a familiarizarnos con esta herramienta, donde teníamos que crear un index.html que generaría una pagina web que imprimiría "HELLO WOLRD".



Una vez hecho esto, hicimos otro ejercicio de prueba para la familiarización con la creación de servlets, en el que tuvimos que crear un archivo llamado web.xml donde creamos el servlet y lo relacionamos con el nombre MyServlet, justo después creamos un archivo java llamado MyServlet.java y ponemos el código necesario para su funcionamiento, el último paso fue compilar y observar que se había realizado con éxito ya que obtuvimos la siguiente página web.



Por último antes de comenzar con el verdadero trabajo, modificamos el fichero conf/context.xml, añadiendo una línea de código específica a este, hicimos esto con el propósito de no tener que reiniciar Tomcat cada vez que modifiquemos el algún fichero .class.

Realización del ejercicio del alquiler de coches

Ahora yendo con la parte gorda del enunciado, teníamos que hacer un servicio web de alquiler de coches en los que realizábamos una petición del coche que alquilábamos y teníamos que guardar la información de la renta en un archivo, para así después cuando iniciemos sesión podamos ver el registro de todos los coches alquilados que hemos realizado, este era el objetivo principal que teníamos que conseguir.

Una parte importante ya nos la daban hecha, que estaba almacenada en un zip llamado carrental.zip, que estaba compuesto de los siguientes archivos: carrental_home.html, carrental_form_new.html, carrental_form_list.html, y dos archivos java parcialmente completados que eran CarRentalNew.java y CarRentalList.java. Descomprimos el zip y los desplazamos a donde Tomcat y

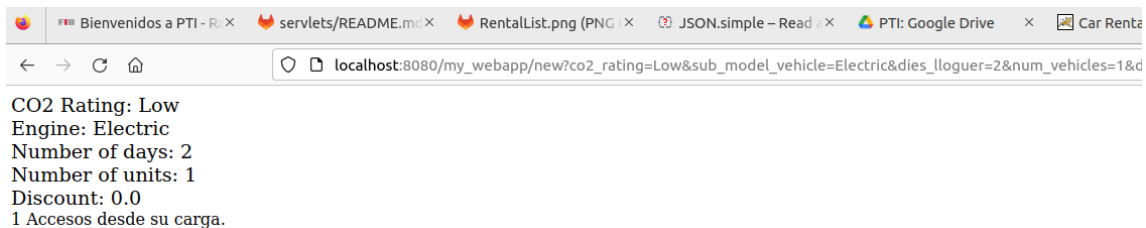
compilamos. Después añadimos a web.xml un par de servlets que estaban relacionados con CarRentalNew y CarRentalList.

Después de todo esto lo que hicimos fue modificar el fichero CarRentalNew.java, para que cada vez que hagamos una nueva solicitud de alquiler de un coche y le demos a submit nos salga una pagina con la información del coche alquilado, esta fue la parte de código que tuvimos que añadir al .java y este fue el resultado que obteníamos al realizar una petición de alquiler:

```
String co2 = req.getParameter("co2_rating");
String engine = req.getParameter("sub_model_vehicle");
String dies = req.getParameter("dies_lloguer");
String unitats = req.getParameter("num_vehicles");
String discount = req.getParameter("descompte");

out.println("<html><big>CO2 Rating: " + co2 + "</big><br>" +
"<html><big>Engine: " + engine + "</big><br>" +
"<html><big>Number of days: " + dies + "</big><br>" +
"<html><big>Number of units: " + unitats + "</big><br>" +
"<html><big>Discount: " + discount + "</big><br>" +
cont + " Accesos desde su carga.</html>");
```

Estas fueron las primeras líneas de código que añadimos y conseguimos visualizar esto al recompilar y dar submit a la hora de hacer una petición.



Lo siguiente que queríamos conseguir era almacenar la información de las peticiones haciendo uso de JSON, para así poder después a la hora de hacer login con admin poder visualizar todo el registro de la renta de coches. Para poder realizar echamos un vistazo a la documentación sobre JSON que había en el enunciado sumado a una búsqueda por nuestra cuenta por internet para ver más documentación de JSON y

tener una mejor idea de su funcionamiento. Nuestra forma de aplicar el uso de esta herramienta fue mediante la implementación de unas líneas de código tanto en el CarRentalNew.java como en el CarRentalList.java. En el primero lo que queríamos es que cada vez que hacíamos una nueva petición esta fuera registrada en un archivo que hiciera la función de una base de datos, y lo conseguimos aplicando las siguientes líneas de código.

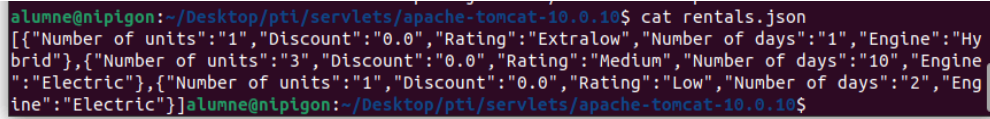
```
JSONObject newrentals = new JSONObject();
newrentals.put("Rating", co2);
newrentals.put("Engine", engine);
newrentals.put("Number of days", dies);
newrentals.put("Number of units", unitats);
newrentals.put("Discount", discount);
//NOU
JSONArray lloguers = null;
try (FileReader reader = new FileReader("rentals.json")) {
    JSONParser parser = new JSONParser();
    lloguers = (JSONArray) parser.parse(reader);
}
catch (IOException | ParseException e) {
    e.printStackTrace();
}

if (lloguers == null) lloguers = new JSONArray();
lloguers.add(newrentals);
try (FileWriter fileWriter = new FileWriter("rentals.json")) {
    fileWriter.write(lloguers.toJSONString());
    fileWriter.flush();
}
catch (IOException e) {
    e.printStackTrace();
}
```

Principalmente lo que hacemos es que cada vez que había una petición de alquiler nos creábamos un JSONObject al que añadíamos todos los parámetros, y donde después leíamos el archivo rental.json y si este estaba vacío creábamos un nuevo JSONArray para inicializarlo y después añadíamos la nueva instancia de petición, y si no estaba vacío simplemente añadíamos la instancia.

*INCISO -> Todos los fragmentos de código que adjuntamos en este archivo también están disponibles en una carpeta comprimida donde esta disponibles todo el enunciado resuelto al completo como son los .xml y los archivos .java.

Después de realizar esto conseguimos guardar la información de las peticiones que realizábamos como se puede observar en la siguiente imagen.

A terminal window with a dark background and light-colored text. The prompt is 'alunne@nipigon:~/Desktop/pti/servlets/apache-tomcat-10.0.10\$'. The command 'cat rentals.json' has been executed, displaying a JSON array of three car rental objects. The first object has 1 unit, 0.0 discount, 'Extralow' rating, 1 day, and 'Hybrid' engine. The second has 3 units, 0.0 discount, 'Medium' rating, 10 days, and 'Electric' engine. The third has 1 unit, 0.0 discount, 'Low' rating, 2 days, and 'Electric' engine.

```
alunne@nipigon:~/Desktop/pti/servlets/apache-tomcat-10.0.10$ cat rentals.json
[{"Number of units": "1", "Discount": "0.0", "Rating": "Extralow", "Number of days": "1", "Engine": "Hybrid"}, {"Number of units": "3", "Discount": "0.0", "Rating": "Medium", "Number of days": "10", "Engine": "Electric"}, {"Number of units": "1", "Discount": "0.0", "Rating": "Low", "Number of days": "2", "Engine": "Electric"}]
```

El siguiente objetivo que teníamos a realizar fue el de que a partir del rentals.json que acumulaba la información de las peticiones, estas poder plasmarlas cuando hacíamos login, para poder realizar esto tuvimos que añadir nuevas líneas de código en el archivo CarRentalList.java. Principalmente lo que hacemos es hacer uso del parser y reader, y crear un bucle for por el que vamos iterando, y cada iteración vamos obteniendo el valor de los parámetros y los printamos para que aparezcan en la pagina web.

```

JSONParser parser = new JSONParser();
try(Reader reader = new FileReader("rentals.json")) {

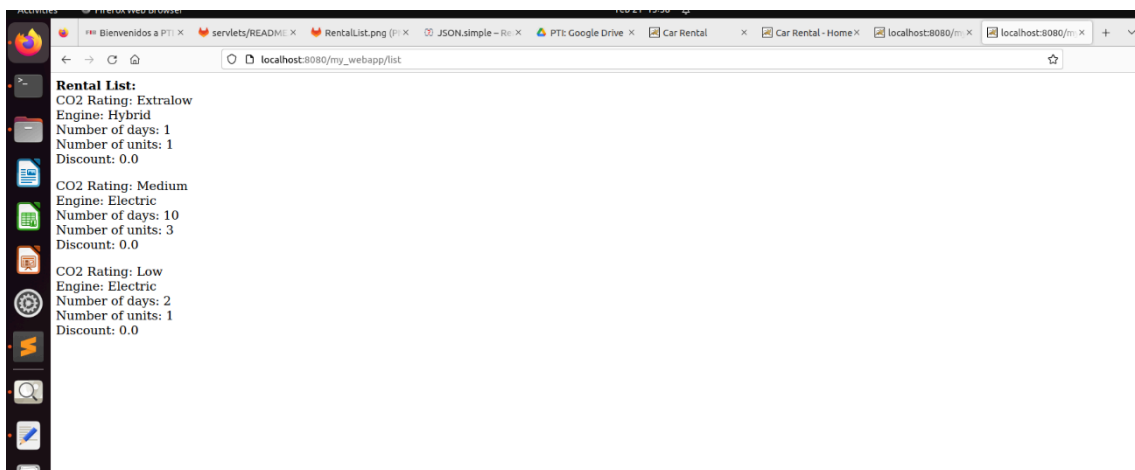
    Object o = null;
    try {
        o = parser.parse(reader);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    JSONArray rentals = (JSONArray) o;

    for (Object obj : rentals) {
        JSONObject lloguer = (JSONObject) obj;
        String num_unitats = (String) lloguer.get("Number of units");
        String descompte = (String) lloguer.get("Discount");
        String rating = (String) lloguer.get("Rating");
        String num_dies = (String) lloguer.get("Number of days");
        String engine = (String) lloguer.get("Engine");
        out.println("<html><big>CO2 Rating: "+ rating + "</big><br>" +
            "<html><big>Engine: "+ engine + "</big><br>" +
            "<html><big>Number of days: "+ num_dies + "</big><br>" +
            "<html><big>Number of units: "+ num_unitats + "</big><br>" +
            "<html><big>Discount: "+ descompte + "</big><br><br></html>");
    }

} catch (IOException e2){
    e2.printStackTrace();
}

```

Recompilamos e iniciamos sesión y obtuvimos el siguiente resultado:



Se puede ver que hay diversas instancias de peticiones de alquiler. Así fue de forma resumida nuestra realización de esta parte del enunciado de la primera práctica.

Configuración SSL/TLS

Ahora nos pusimos a realizar la última parte de la práctica que consistía en hacer una configuración SSL/TLS para fortalecer la página web y no hacerla pública. Lo primero que hicimos fue ejecutar el comando especificado en el enunciado y introduciendo la contraseña que usaremos, después modificamos el fichero `sconf/server.xml` que teníamos y añadimos la siguiente parte de código que estaba especificado el enunciado pero que además tuvimos que añadir una línea más ya que estaba incompleto, que fue la de `"certificateKeystorePassword= "JoanLuis""`.

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"

sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"

port="8443"

maxThreads="150"

SSLEnabled="true">

<SSLHostConfig>

  <Certificate

    certificateKeystoreFile="${user.home}/.keystore"

    certificateKeystorePassword= "JoanLuis"

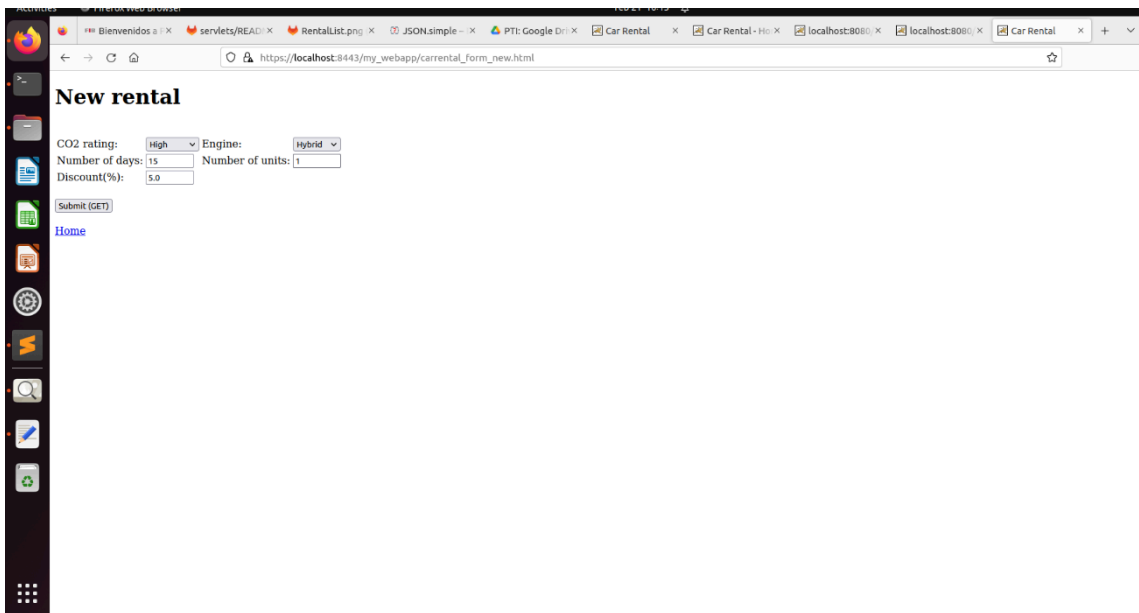
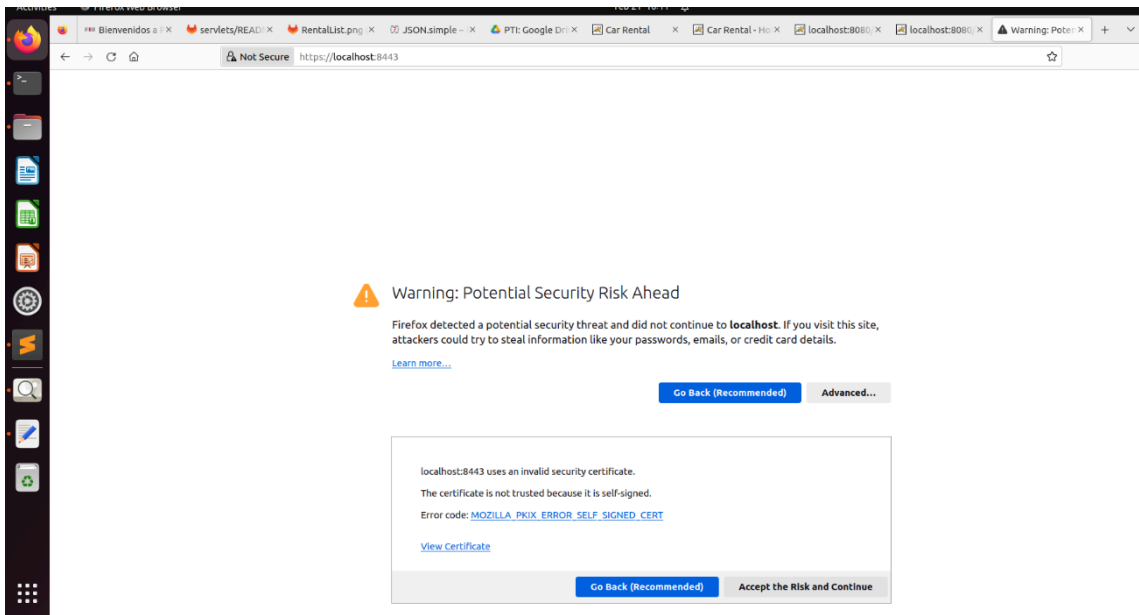
    type="RSA"

  />

</SSLHostConfig>

</Connector>
```

Después de realizar todo esto reiniciamos Tomcat y pudimos observar que funcionaba correctamente ya que al ir a la página nos aparecía lo siguiente:



Dockerització

Primero de todo hemos instalado el docker en nuestra máquina virtual, una vez instalada y configurada hemos probado su funcionamiento mediante un docker run hello-world. Una vez comprobado hemos creado y editado el fichero Dockerfile para más tarde iniciar la imagen y comprobar que efectivamente la aplicación corría correctamente.

