Extensió d'un servidor amb servlets

Projecte de Tecnologies d'Informació

1. Introducció a tomcat i servlets

Apache Tomcat, es un servidor web de programari liure desenvolupat per Apache. Tomcat implementa diferents especificacions de Java EE, entre les quals trobem els Java Servlets. Tomcat està format per Catalina (com a contenidor de servlets), Coyote (com a connector HTTP) i Jasper (un motor JSP).

Ademés, amb la versió de tomcat 7 s'hi han afegit 3 components. El cluster (per gestionar aplicacions grans), una funcionalitat d'alta disponibilitat per facilitar la gestió d'actualitzacions del sistema (com per exemple versions noves). També s'ha afegit millores a aplicacions web que incorporen suport per distribució en diferents entorns.

Un servlet de java és un programa que exten les funcionalitats d'un servidor. Tot i que els servlets poden respondre a qualsevol tipus de peticions, normalment s'implementen per aplicacions en web servers. Aquests web servers acostumen a ser la contrapartida de java d'altres technologies web de contingut dinàmic com PHP.

Un objecte Servlet pot rebre una invocació i generar una resposta en funció de les dades de la invocació, de l'estat del mateix sistema i les dades a què pugui accedir. El paquet bàsic de servlet defineix objectes Java que gestionen peticions I respostes fetes per servlets, així com objectes que informen dels paràmetres de configuració del servlet i l'entorn d'execució.

El cicle de vida d'un servlet consta de 3 fases; en primer lloc, cal inicialitzar el servlet (executant el mètode init del servlet). El procés d'inicialització s'ha de completar abans de poder gestionar peticions dels clients. Després, el servlet pot interactuar amb els clients, és a dir, respondre a les peticions. Aquestes peticions seran ateses per la mateixa instància del servlet. Per tant, s'ha de vigilar al accedir a variables compartides, ja que podrien donar-se problemes de sincronització entre requeriments simultanis.

2. Instal·lació i configuració de Java / tomcat / servlets

Per instal·lar i configurar tomcat hem seguit els passos que presentem a continuació. Disposem de privilegis de superusuari (root), per tant la instal·lació és bastant senzilla.

La majoria de passos que hem seguit es troben al següent enllaç, proporcionat per el professor:
https://github.com/rtous/pti/tree/master/p2_servlets

En primer lloc es va executar la comanda java version per tal de verure si tenim el java instal·lat a la màquina. Tal i com podem observar en la imatge, el java no estava instal·lat, per tant vam procedir a la instal·lació d'aquest.

```
root@a5s111pc35:/home/alumne# java -version
The program 'java' can be found in the following packages:
 * default-jre
 * gcj-4.8-jre-headless
 * openjdk-7-jre-headless
 * gcj-4.6-jre-headless
 * openjdk-6-jre-headless
Try: apt-get install <selected package>
```

D'aquesta manera, vam afegir el repositori de java en el apt-get i a continuació vam fer un apt-get update per tal de actualitzar les bases de dades de paquets amb el nou repositori que haviem acabat d'afegir. A continuació, ja vam poder executar el "apt-get install" dels packages , "oracle-java8-installer" I "oracle-java8-set-default". Finalment vam crear un soft-link per tal de posar el nou java8 com a el nostre java per defecte.
Així, al tornar a executar la comanda "java -version", ja vam podern observar que el java s'havia instal·lat correctament, exactament la versió 1.8.

```
root@a5s111pc35:/home/alumne# java -version
java version "1.8.0_72"
Java(TM) SE Runtime Environment (build 1.8.0_72-b15)
Java HotSpot(TM) Server VM (build 25.72-b15, mixed mode)
```

El pas següent va ser instal·lar el servidor d'aplicacions tomcat7. Per tal d'instal·lar aquest va, executar les següents comandes:

$sudo apt-get install tomcat7
$sudo apt-get install tomcat7-docs tomcat7-admin

Un cop executades aquestes comandes ja vam tenir instal·lat el servidor tomcat I la seva documentació I les eines d'administrador. Aleshores vam passar a executar el servei I vam poder visualitzar l apàgina localhost:8080

It works !

A continuació, vam passar a la creació de la aplicació web. Per tal de crear aquesta vam crear una carpeta dins el directori d'aplicacions (webapps) anomenada my_webapp. Aleshores vam crear el nostre primer fitxer html per servir-lo. Un cop reiniciat el servidor ja vam poder observar aquesta pàgina.

Per tal de configurar els servlets, aquesta introducció als servlets primer vam crear un nou directori en el directori de la nostra aplicació web(my_webapp).

A continucació vam crear el fitxer web.xml en el qual el que vam fer va ser indicar el nom dels servlets que teniem (en el cas d'exemple solament 1), I a més vam indicar el mapeix de quins servlets es tenien que executar en quines urls. Després vam crear un directori , el mypackage on guardar totes les classes java que voliem fer.I vam crear el primer servlet , el MyServlet.java.

3. Objectius de la pràtica i implementació

L'objectiu d'aquesta pràctica és crear una pàgina web de lloguer de cotxes. Bàsicament tindrà dues funcionalitats:

1 - Fer una nova petició, consistent en un formulari per introduir una nova comanda. Els camps del formulari inclouran el fabricant del cotxe, el model, el número de dies durant els quals es vol llogar el vehicle I finalment el nombre d'unitats que es volen llogar. Si les dades són correctes, es retornarà el preu total del lloguer, ademés de les dades de la petició de lloguer.

2 – Obtenir la llista de tots els lloguers. Un formulari en el qual s'ha d'introduir una contrasenya (que teòricament només coneix l'administrador) permetrà visualitzar una llista amb totes les comandes guardades. Per fer-ho és necessari guardar les comandes que es produeixen en un fitxer.

Els fitxers carrental_home.html, carrental_form_new.html i carrental_form_list.html ja formen part de l'enunciat, de manera que no és necessari que els programem. Per tant, ens centrarem en programar els servlets que implementen cadascuna de les dues funcionalitats descrites. Per tant, tindrem un servlet CarRentalNew.java corresponent a la primera funcionalitat i un segon, CarRentalList.java, corresponent a la segona funcionalitat.

1 - Nova petició: CarRentalNew.java

Tal i com hem dit, la primera funcionalitat corresponent a l'acció de creació d' una nova comanda serà definida a CarRentalNew.java. A continuació presentem el codi i la seva justificació. En primer lloc, utilitzarem una funció per generar el contingut: print_orders(), que imprimeix el resum de la comanda, I les dades de la mateixa. També incorpora un càlcul del preu, amb el descoopmte aplicat si fós convenient.

```
public void print_orders(HttpServletResponse res) throws IOException  {
  PrintWriter out = res.getWriter();
  BufferedReader br = null;
    try {
        String sCurrentLine;
        br = new BufferedReader(new FileReader("/var/lib/tomcat7/webapps/data.json"));
        out.println("<H1><center><u> Historial de comandes </u></center><H1>");
        int comanda = 1;
        while ((sCurrentLine = br.readLine()) != null) {
          JSONParser parser = new JSONParser();
          Object obj = parser.parse(sCurrentLine);
          JSONObject json = (JSONObject) obj;
          out.println("<H3> <strong><i> >> Comanda "+comanda+" </i></strong> <H3>"+
                      "<DIV STYLE=\"background-color: lightyellow;border:1px solid;border-radius: 10px;width:
          out.println("<H4> Model: " +json.get("model")+ "  |  Combustible: " +json.get("sub_model_vehicle")+
          out.println("<br>");
          //out.println("<html><big>"+json.get("sub_model_vehicle")+"</html></big>");
          ++comanda;
        }
      } catch (Exception e) {
                out.println("<html><big>hola</html></big>");
            e.printStackTrace();
      } finally {
          try {
              if (br != null)br.close();
          } catch (IOException ex) {
              ex.printStackTrace();
          }
      }

}
```

Codi corresponent a la funció print_orders().

Com es pot observar, a la funció print_order() també hem afegit un DIV css per fer el disseny més atractiu.

Además, també és necessari comprovar que els camps que introdueix l'usuari siguin correctes. És una bona pràctica assumir que no ho seran per evitar errors. Realitzem la comprovació a la part del codi mostrat a continuació. En cas de que un camp no satisfaci les condicions de correctesa, s'utilitza la s'informarà a l'usuari de la situació per a que retrocedeixi.

Primer obtenim els valors corresponents als camps, i després comprovem que siguin correctes.

```
//parametres
//String nombre = req.getParameter("name");
int dies = Integer.parseInt(req.getParameter("dies_lloguer"));
String model = req.getParameter("model_vehicle");
int num_model = Integer.parseInt(req.getParameter("model_vehicle"));
double descompte = Double.parseDouble(req.getParameter("descompte"));
String sub_model_vehicle = req.getParameter("sub_model_vehicle");
int num_vehicles = Integer.parseInt(req.getParameter("num_vehicles"));
double preu_total = ((100-descompte)/100)*num_vehicles*dies*num_model;

//obtenir model en text
String model_real;
if (model.equals("54")) model_real = "Economic";
else if (model.equals("71")) model_real = "Semi-luxe";
else if (model.equals("82")) model_real = "Luxe";
else model_real = "Limusina"; //139
```

Obtenció de paràmetres.

Un cop hem obtingut els paràmetres, comprovem que siguin correctes. En cas de no ser-ho informem a l'usuari mitjançant la impressió d'un error.

```
//comprovacio
String error = "no error";
if (dies < 1) error = "Numero de dies incorrecte";
if (descompte >= 100.0) error = "Descompte incorrecte";
if (num_vehicles < 1) error = "Numero de vehicles incorrecte";

if (!error.equals("no error")) out.println("<html><big>" + error + "</big></html>");
```

Comprovació de paràmetres

Un cop sabem que els paràmetres són correctes, els guardem. En aquest cas utilitzarem JSON, per tant creem un objecte JSON en el que posem tots els paràmetres.

```
else {
    JSONObject obj = new JSONObject();
    obj.put("dies", dies);
    obj.put("model", model_real);
    obj.put("descompte", descompte);
    obj.put("sub_model_vehicle", sub_model_vehicle);
    obj.put("num_vehicles", num_vehicles);
    obj.put("preu",String.valueOf(preu_total));
```

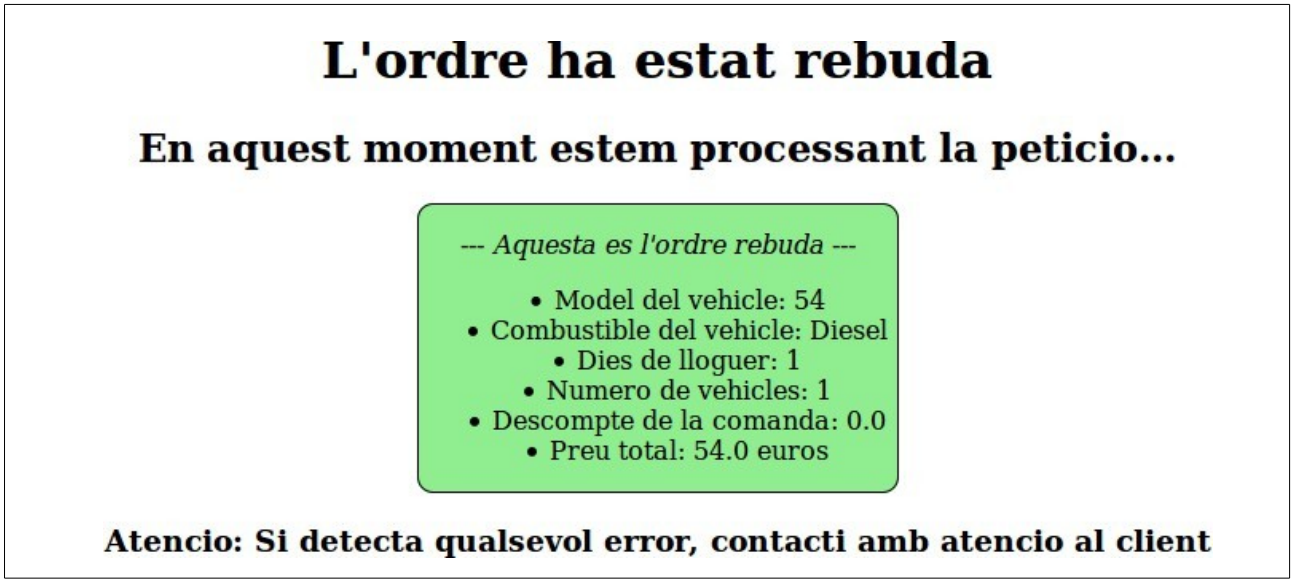
Creació i inicialització del objecte JSON.

Després, creeem el fitxer JSON en cas que no existeixi I guardem l'objecte al fitxer. Per tal de fer més fàcil la lectura, hi guardem un senyal de fi de línia "\n". Això ens permet llegir per línies a l'hora de llistar les comandes. Donat que poden sorgir problemes que generin excepcions, emboliquem les crides en un bloc try{}.

```
try {
    FileWriter file = new FileWriter("/var/lib/tomcat7/webapps/data.json", true);
    file.write(obj.toJSONString());
    file.write("\n");
    file.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Creació del fitxer, I escriptura de l'objecte JSON.

Un cop s'ha completat el procés, l'usuari pot veure una resum de la comanda amb el preu de la mateixa.



Confirmació i resum de la comanda

2 - Llistar peticions: CarRentalList.java

De la mateixa manera que en el cas anterior, tenim una funció print_orders() que imprimeix les comandes i els hi aplica un estil CSS.
Per fer-ho, llegeix les línies del fitxer "data.json" i les converteix en objectes JSON. Un cop fet això, obté els camps de l'objecte i imprimeix les dades de la comanda.

```
public void print_orders(HttpServletResponse res) throws IOException  {
  PrintWriter out = res.getWriter();
  BufferedReader br = null;
    try {
        String sCurrentLine;
        br = new BufferedReader(new FileReader("/var/lib/tomcat7/webapps/data.json"));
        out.println("<H1><center><u> Historial de comandes </u></center><H1>");
        int comanda = 1;
        while ((sCurrentLine = br.readLine()) != null) {
          JSONParser parser = new JSONParser();
          Object obj = parser.parse(sCurrentLine);
          JSONObject json = (JSONObject) obj;
          out.println("<H3> <strong><i> >> Comanda "+comanda+" </i></strong> <H3>"+
                      "<DIV STYLE=\"background-color: lightyellow;border:1px solid;border-radius: 10px;width:
          out.println("<H4> Model: " +json.get("model")+ "  |  Combustible: " +json.get("sub_model_vehicle")+
          out.println("<br>");
          //out.println("<html><big>"+json.get("sub_model_vehicle")+"</html></big>");
          ++comanda;
        }
      } catch (Exception e) {
                out.println("<html><big>hola</html></big>");
            e.printStackTrace();
      } finally {
          try {
              if (br != null)br.close();
          } catch (IOException ex) {
              ex.printStackTrace();
          }
      }
```

Funció print_orders() per imprimir les comandes.

Abans d'imprimir les comandes cal comprovar les dades. Per tant, s'obtenen els camps del formulari. Si algun dels camps no hi és o bé la contrasenya és incorrecta, s'informa a l'usuari a través d'un error. En cas de que la contraseña (apache) sigui correcta, es crida a la funció print_orders().

```
String userid = req.getParameter("userid");
String password = req.getParameter("password");

String error = "no error";
if (userid.equals("")) error = "Error: cal introduir un usuari";
else if (password.equals("")) error = "Error: cal introduir un password";

if (!error.equals("no error")) {
    out.println("<html><big>" + error + "</big></html>");
}
else {
    if (password.equals("apache")) print_orders(res);
    else out.println("<html><big> Contrasenya incorrecta </big></html>");
}
```

Comprovació dels camps del formulari

Si la contrasenya és correcte, l'usuari podrà veure un llistat de les comandes emmagatzemades:

Vista del resum de les comandes.

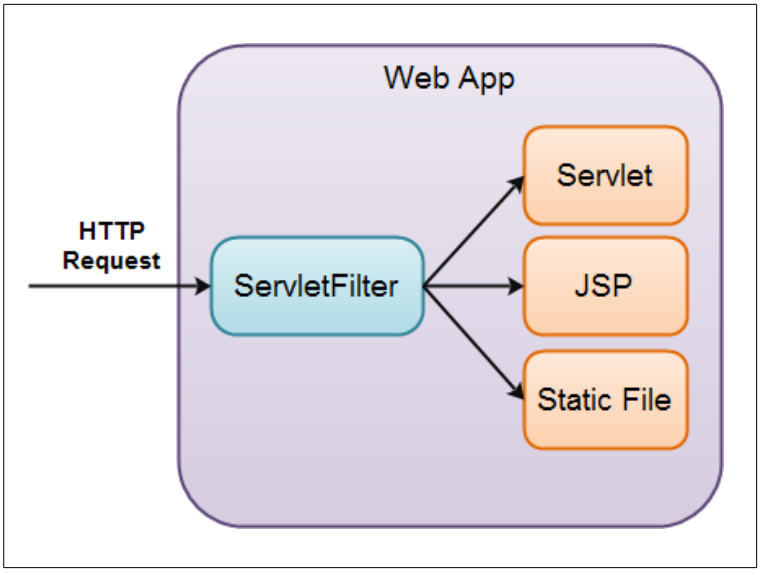
4. Servlets: avantatges i inconvenients

S'ha de tenir molt en compte que quan el servidor de pàgines carrega i executa un servlet , el manté viu a memòria en el seu propi procès per tota la vida del servidor.

És per això que la seva utilització comporta certes avantatges i desavantatges. Un dels avantatges que podem observar és que un cop ja tenim un servlet carregat, si un altre usuari demana el mateix la resposta és farà de manera ràpida. Però a la vegada també pot ser que una aplicació web tingui molts servlets i que n'hi hagi alguns que s'utilitzin pocs cops ,el que provocaria que tinguem carregat servlets en memòria gastant recursos , que gairebé no utilitzarem.

També cal nombrar que es poden utilitzar jsp(java server pages) , que són pàgines html que tenen codi java encastat i HTML generat automàticament, és a dir , són una ampliació dels servlets.
En el cas de JSP existeix una avantatge i és que el codi queda molt més ben estructurat , ja que en l'arxiu combinem el html estructurat amb codi java. Al contrari que passa en el cas del servlets , on el codi html és molt poc comprensible, ja que al igual que en el cas dels CGIs , en el codi java es posa el codi HTML.

Un desavantatge molt clara del JSP, és la visibilitat, és a dir, des de qualsevol explorador a l'hora de visualitzar la pàgina, qualsevol usuari podrà observar el codi java que estem utilitzant i podrà tenir més intuïció de com intentat «hackejar». Altrament, en el cas del servlets el codi java no és visible per als usuaris, per tant, d'aquesta forma augmentem la seguretat.

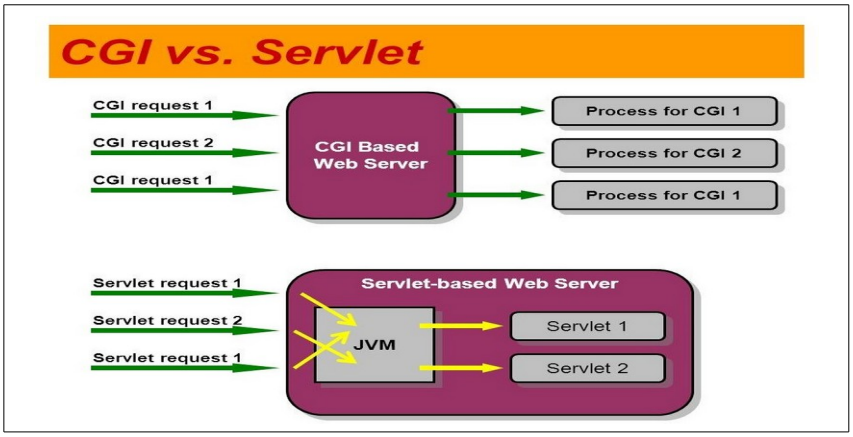


Esquema del procés operatiu d'un servlet

5. Comparació de CGIs amb Servlets

En la imatge podem observar la gran diferència entre els CGIs i els servlets. En el cas dels CGIs és creava un procés a banda del servidor per tractar les peticions.

En canvi en el cas dels servlets , aquests es creen en el servidor , també gastant memòria i es mantenen vius. I com podem veure en la imatge, en el cas de rebre una petició de un servlet que ja el tenim actiu no es crea un altra vegada sinó que s'aprofita.



Comparació de Servlets amb CGIs

6. Tomcat com a servidor d'aplicacions

Tal i com podem observar en la gràfica tomcat és el servidor d'aplicacions més utilitzat en diferència. Tenint una quota de mercat del 58,66& , tot seguit del 15,7% de JBOSS i el 9,93% de WebLogic.

Una de les principals diferencies entre Tomcat i JBOSS és que tomcat solament implementa els servlets i jsp.En canvi JBOSS és un servidor d'aplicacions java complet(ho implementa tot(EJB,etc)).