4. Recursos de programación con JSP.

1. Introducción.

En el presente capítulo veremos algunos recursos extra de programación de aplicaciones web en plataformas JavaEE. Dichos recursos irán orientados a realizar ciertas tareas específicas en unos casos y a añadir conectividad con otras plataformas en otros, aportando siempre valor añadido a nuestras aplicaciones.

Veremos estos recursos de una forma práctica, realizando para cada uno de ellos una breve introducción a la que seguirán ejemplos de código comentados, que serán fácilmente reutilizables en cualquier aplicación.

2. Subir ficheros al servidor.

Es muy común en aplicaciones web, realizar subidas de ficheros/documentos desde el cliente al servidor, ya sea simplemente para almacenarlos o para procesarlos. Ésta es una técnica estándar basada en el uso de formularios HTML con codificación "*multipart/form-data*" y campos "<*input type='file'*>". Lo que se diferencia entre unas plataformas y otras, es la forma en que el servidor trata los datos recibidos. En particular, en la plataforma JavaEE, los datos estarán accesibles a través del objeto *request*, usando el método *getPart()*. Hay que tener en cuenta que cuando se usa la codificación "*multipart/form.data*", todos los parámetros procedentes de los campos del formulario se recibirán en el servidor como partes que contendrán un nombre y una secuencia de datos binarios. El método *getPart()* del objeto *request* acepta como argumento una cadena con el nombre del parámetro a leer, y nos devuelve un objeto de tipo *Part*, que nos permite entre otras cosas obtener una secuencia de entrada para leer sus datos – *getInputStream()* -, consultar la longitud de los datos – *getSize()* - , consultar el tipo de contenido – *getContentType()* - o salvar los datos directamente en un fichero – *write()* -.

A continuación veremos un ejemplo de código diseñado para subir fotos de perfil de los usuarios al servidor y almacenarlas en un directorio específico. Primero veremos un fragmento de página JSP con el formulario HTML y a continuación el servlet que recibe los datos del formulario y los procesa. En ésta ocasión se ha usado un servlet ya que se realiza el procesamiento de la información sin generar ninguna respuesta. En su lugar, se reenvía el control a la página de la que procede.

cambiarFoto.jsp:

```
<form id="formulario" action="subirFoto" method="post"
enctype="multipart/form-data">
   <input type="file" name="foto" size="10"
onchange="actualizar();">
   <input type="hidden" name="nombreFichero" value="empleado$
{empleado.id}">
</form>
```

SubirFoto.java:

```
public class subirFoto extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)

    throws ServletException, IOException {

        try {

            String error = null;

            String nombre = "";

            String extension = "";

            BufferedReader lectorNombre = new BufferedReader(new
InputStreamReader(request.getPart("nombreFichero").getInputStream()));

            nombre = lectorNombre.readLine();

            lectorNombre.close();

            Part datosSubidos = request.getPart("foto");

            if (datosSubidos == null) {  // No se ha subido el fichero

                error = "No se ha recibido la imagen";

            }

            else

                if (datosSubidos.getSize() > 100*1024) {  // Fichero
demasiado grande

                    error = "No se permiten ficheros superiores a 100Kb";

                }

                else

                    if (datosSubidos.getContentType().indexOf("image")==-
1) {  // El fichero no es una imagen

                        error = "El fichero recibido no es una imagen
válida";

                    }

                    else {

                        String tipoContenido =
datosSubidos.getContentType();

                        int posicion = tipoContenido.indexOf("/");

                        extension = tipoContenido.substring(posicion + 1);

                    }

            if (error==null) {

                nombre = nombre+"."+extension;

                String contexto =
request.getServletContext().getRealPath("fotos/" + nombre);
```

```
                FileOutputStream fichero = new FileOutputStream(contexto);
                InputStream contenido = datosSubidos.getInputStream();
                byte[] bytes = new byte[2048];
                while (contenido.available()>0) {
                    int longitud = contenido.read(bytes);
                    fichero.write(bytes, 0, longitud);
                }
                fichero.close();
            }
            request.setAttribute("nombreFichero", nombre);

request.getRequestDispatcher(response.encodeURL("cambiarFoto.jsp" +
parametros)).forward(request, response);
        }
        catch (Exception e) {
            e.printStackTrace(out);
        } finally {
            out.close();
        }
    }
```

Además debemos añadir a la configuración del servlet una sección de configuración multipart en el descriptor de despliegue, web.xml, o bien una anotación *@MultipartConfig* en el propio código fuente del servlet. La configuración multipart soporta cuatro atributos opcionales:

- *location*: la ruta absoluta a un directorio del sistema de ficheros para almacenar temporalmente los datos subidos mientras son procesados o cuando su tamaño supera el valor indicado en el atributo *fileSizeThreshold*.
- *fileSizeThreshold*: el tamaño de fichero en bytes a partir del cual el fichero será almacenado temporalmente en disco. El valor por defecto es 0.
- *maxFileSize*: el tamaño máximo permitido para los ficheros subidos. Si el tamaño de un fichero supera éste valor, el servidor JEE lanzará una excepción de tipo *IllegalStateException*. Por defecto el tamaño permitido es ilimitado.
- *maxRequestSize*: el tamaño máximo permitido para la petición multipart. Si el tamaño de la petición es superior a éste valor, el servidor JEE lanzará una excepción. Por defecto el tamaño permitido para la petición es ilimitado.

web.xml:

```
<multipart-config>
    <location>/tmp</location>
    <max-file-size>20848820</max-file-size>
    <max-request-size>418018841</max-request-size>
    <file-size-threshold>1048576</file-size-threshold>
</multipart-config>
```

3. Enviar e-mail.

Una tarea bastante habitual en aplicaciones web es la de enviar emails, por ejemplo para confirmar un pedido o para enviar un código de activación. Aquí podemos sacar ventaja de las numerosas APIs Java, y en particular de la API JavaMail.

Veremos un ejemplo estructurado en dos clases, Email.java, que describe un mensaje de correo electrónico con sus componentes, y Utilidades.java, que contiene el método enviarEmail(), que realiza el envío usando una cuenta de correo previamente creada en un servidor.

Email.java:

```
public class Email {
    private String from = "";
    private String to = "";
    private String subject = "";
    private String text = "";

    public String getFrom() {
        return from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getTo() {
        return to;
    }
}
```

Email.java(continuación):

```
public void setTo(String to) {
        this.to = to;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }


}
```

```
import javax.mail.Message.RecipientType;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

Utilidades.java:Utilidades.java(continuación):

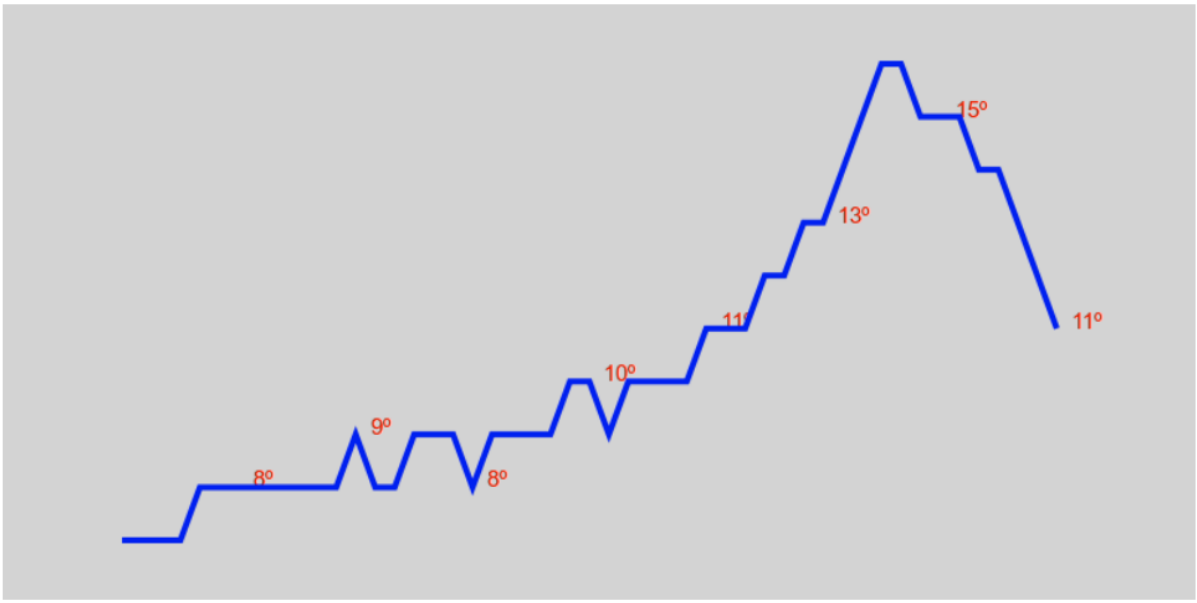
```
Public class Utilidades {
    public void setEnviarEmail(Email email){
        Properties p = new Properties();
        // Servidor smtp de correo
        p.setProperty("mail.smtp.host", "smtp.gmail.com");
        // Usar TLS
        p.setProperty("mail.smtp.starttls.enable", "true");
        // puerto del servidor smtp
        p.setProperty("mail.smtp.port", "587");
        // Usuario smtp
        p.setProperty("mail.smtp.user", email.getFrom());
        // Autenticación requerida
        p.setProperty("mail.smtp.auth", "true");
        // Obtenemos la sesión
        Session sesion = Session.getDefaultInstance(p);
        sesion.setDebug(false);
        // Creamos el mensaje
        MimeMessage mensaje = new MimeMessage(sesion);
        // Y establecemos sus propiedades
        try {
            mensaje.setFrom(new InternetAddress(email.getFrom()));
            mensaje.addRecipient(RecipientType.TO, new
InternetAddress(email.getTo()));
            mensaje.setSubject(email.getSubject());
            mensaje.setText(email.getText());
            // Enviamos el mensaje
            Transport t = sesion.getTransport("smtp");
            // Para conectarnos usamos usuario y password
            t.connect(email.getFrom(), "password");
            t.sendMessage(mensaje, mensaje.getAllRecipients());
        } catch (MessagingException e) {
            emailError = e.getMessage();
        }
    }
   ...
}
```

4. Representar gráficos.

Para representar gráficos en una aplicación web tenemos varias aproximaciones posibles. La primera consiste en generar los gráficos en un fichero de imagen por medio de alguna biblioteca específica y desplegar dicho fichero haciendo referencia a él desde nuestra página con un elemento <img>. Esta es una tecnica bastante costosa para el servidor en términos de procesamiento y de almacenamiento, aunque tiene como ventaja la independencia de la plataforma del cliente.

Hay otras soluciones basadas en generar código ejecutable en el cliente para que sea éste quien realice la gráfica.

En primer lugar veremos un ejemplo en el que se usa un elemento canvas (HTML 5) para dibujar sobre él usando funciones javascript a partir de un arry de datos generados en el servidor. En el presente ejemplo se dibuja una gráfica con datos de temperaturas tomadas en un periodo determinado, pero en general, podríamos realizar cualquier dibujo basado en líneas, polígonos, figuras geométricas, etc



grafica.jsp:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<jsp:useBean id="datos" class="beans.Datos" scope="page"/>
<!DOCTYPE html>
<html>
    <head>
        <script>
            function dibujar_grafica() {
                var datos = [
                    <c:forEach items="${datos.datos}" var="dato"
varStatus="status" >
                                ${dato.temperatura}
                                <c:if test="${!status.last}">
                                    ,
                                </c:if>
                    </c:forEach>
                ];
                var canvas = document.getElementById('grafica');
                var gr = canvas.getContext('2d');
                gr.font='15px Arial';
                gr.beginPath();
                // obtenemos las dimensiones del canvas
                ancho = canvas.clientWidth;
                alto = canvas.clientHeight;
                // Dibujamos dejando un margen del 10%
                x_derecha = ancho * 0.1;
                y_abajo = alto * 0.9;
                // Buscamos el maximo de las temperaturas
                min =  100000;
                max = -100000;
                for (i=0;i<datos.length;i++) {
                    if (datos[i]>max) {
                        max = datos[i];
                    }
                    if (datos[i]<min) {
                        min = datos[i];
                    }
```

grafica.jsp (continuación):

```
                }
                // Calculamos la escala vertical y el paso horizontal
                escala_y = alto*0.8/(max-min);
                paso_x = ancho*0.8/datos.length;
                gr.fillStyle='#F02000';
                // Nos posicionamos en el primer punto de la grafica
                gr.moveTo(x_derecha, y_abajo-(datos[0]-
min)*escala_y);
                x = x_derecha;
                // Recorremos el resto de los datos dibujando
segmentos de linea
                for(i=1;i<datos.length;i++) {
                    x+=paso_x;
                    gr.lineTo(x,y_abajo-(datos[i]-min)*escala_y);
                    // Cada 6 puntos mostramos el valor
                    if(i%6==0) {
                        gr.fillText(datos[i]+'º',x+10,y_abajo-
(datos[i]-min)*escala_y);
                    }
                }
                gr.strokeStyle='#0020F0';
                gr.lineWidth = 4;
                // Terminamos la grafica
                gr.stroke();
            }
        </script>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Temperaturas</title>
    </head>
    <body onload="dibujar_grafica()">
        <h3>Temperaturas</h3>

        <canvas id="grafica" height="400" width="800"
style="background-color: lightgrey" ></canvas>
    </body>
</html>
```

La página grafica.jsp usa un bean de la clase beans.Datos que contiene una propiedad llamada datos que le proporciona una colección con los datos de temperatura. La parte más importante desde el punto de vista del procesamiento en el servidor es la que se encuentra entre "var datos = [" y "];". En esas líneas generamos el código javascript para crear un array con los datos obtenidos del bean mediante un bucle forEach. El resto de la página es el código javascript que dibuja la gráfica a partir de ese array – y que también podría encontrarse separado en un fichero .js – y el elemento canvas que contendrá la gráfica.

Otra solución es usar una biblioteca externa, como por ejemplo, HIGHCHARTS (www.highcharts.com). Se trata de una API javascript para el trazado de gráficas de datos, que ofrece multitud de tipos de gráficas y de opciones. Aunque no tenemos tanta flexibilidad para dibujar como en el caso anterior, con el uso de esta API conseguimos mucha potencia con poco esfuerzo.

En el siguiente ejemplo se muestra una gráfica de los resultados electorales:

estadisticas.jsp:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<jsp:useBean id="datos" class="entidades.Datos" scope="session"/>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.j
s"></script>
```

```
        <script
src="https://code.highcharts.com/highcharts.js"></script>
        <script src="https://code.highcharts.com/highcharts-
3d.js"></script>
        <title>Estadisticas</title>
    </head>
    <body>
        <h1>Estadisticas</h1>
        <div id="container" style="min-height: 400px; min-width:
600px;">
        <script>
            $(function () {
                $('#container').highcharts({
                    chart: {
                        type: 'pie',
                        options3d: {
                            enabled: true,
                            alpha: 45
                        }
                    },
                    title: {
                        text: 'Resultado de las elecciones'
                    },
                    subtitle: {
                        text: 'Votos recibidos por cada partido'
                    },
                    plotOptions: {
                        pie: {
                            innerSize: 100,
                            depth: 45,
                            startAngle: -90,
                            endAngle: 90
                        }
                    },
```

estadisticas.jsp (continuación):

estadisticas.jsp(continuación):

```
                    series: [{
                        name: 'Votos',
                        data: [
                <c:forEach items="${datos.partidos}" var="partido"
varStatus="status">
                                            ['<c:out value="$
{partido.nombre}" />', <c:out value="${partido.votos}"/>]  <c:out
value="${status.last?'':','}"/>
                <c:forEach>
                        ]
                    }],
                    colors: [
                <c:forEach items="${datos.partidos}" var="partido"
varStatus="status">
                    '<c:out value="${partido.color}"/>' <c:out
value="${status.last?'':','}"/>
                <c:forEach>
                    ]
                });
            });
        </script>
    </body>
</html>
```

En el código de la página estadisticas.jsp vemos como los datos se extraen de un bean de la clase entidades.Datos. El código javascript se ha obtenido de las plantillas que se pueden encontrar en la página web de HIGHCHARTS, y dentro de ese código se han insertado los datos en las secciones series y colors, de forma similar a como se hizo en el ejemplo anterior.

En algunas ocasiones nos puede bastar con usar determinados elementos html configurados adecuadamente para generar gráficos sencillos. Sirva de ejemplo el siguiente fragmento donde se muestran también los votos de los partidos, pero esta vez mediante barras obtenidas con elementos a los que se les ha dado color de fondo y dimensiones adecuadas en función de los datos que se quiere representar.

Partido	Nº de Votos	Porcentaje de votos	Gráfica
PoPo	2	12%	
PASAO	3	19%	
Cuidadnos	3	19%	
Potemos	3	19%	
ui	2	12%	
En Blanco	3	19%	

estadisticas.jsp(fragmento):

```
        <table>
            <tr><th>Partido</th><th>Nº de Votos</th><th>Porcentaje de
votos</th><th>Gráfica</th></tr>
            <c:forEach items="${datos.partidosJPA}" var="partido">
                <tr style="color: ${partido.color};">
                    <td>${partido.nombre}</td>
                    <td style="text-align: right;">$
{partido.votos}</td>
                    <td style="text-align: right;"><fmt:formatNumber
value="${partido.votos/datos.totalVotos}" type="percent" /></td>
                    <td><div title="${partido.nombre}"
style="background-color: ${partido.color}; width: $
{partido.votos*20}px;height: 20px;"></div></td>
                </tr>
            </c:forEach>
        </table>
```

5. Ajax.

Ajax (Asynchronous Javascript and XML), es como su nombre indica, una tecnología de cliente. Consiste en realizar peticiones asíncronas (en cualquier momento) desde el código JavaScript (o Jquery) de una página. La respuesta a dicha petición será procesada también por el código javascript. La respuesta puede ser código HTML para insertarlo en algún elemento de la página, XML o JSON que contenga datos que serán procesados en el cliente, etc. Uno de los usos más comunes de Ajax, consiste precisamente en refrescar algún elemento de la página a partir del código HTML recibido. Las peticiones Ajax se pueden configurar también para que envíen datos en forma de parámetros de la petición, cookies, etc.

Desde el punto de vista del servidor, lo único que hay que tener en cuenta es que el módulo web que reciba una petición Ajax procese los parámetros adecuados, si los hay, y que genere el contenido esperado por la petición. Por ejemplo, si se trata e refrescar un elemento de una página, sólo habrá que enviar el contenido de ese elemento, y no una página completa.

A continuación veremos un ejemplo de una página que genera un listado de empleados y permite filtrar los empleados mediante Ajax. En la página listarEmpleados.jsp se incluye una llamada Ajax que se produce cuando se pulsa el botón “Filtrar” y que envía como parámetro la cadena introducida por el usuario en el campo “filtro”. La llamada realizará una petición al módulo filtrarEmpleados.jsp, que a su vez, usará el parámetro recibido para filtrar la consulta de empleados en la base de datos y devolver una tabla HTML con la información de los empleados obtenidos,

listarEmpleados.jsp(fragmento):

```
  <html>
    <head>
         <title>Empleados</title>
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js
"></script>
        <script>
            function filtrar() {
                var filtro = document.getElementById('filtro').value;

                $.ajax({
                  method: "POST",
                  url: "filtrarEmpleados.jsp",
                  data: { filtro: filtro }
                })
                  .done(function( listado ) {
                    $("#listadoEmpleados").html(listado);
                });
            }
        </script>
    </head>
    <body>
        <h1>Empleados</h1>
        <div style="background-color: beige;padding: 10px;">
            <form action="filtrarEmpleados.jsp"
onsubmit="filtrar();return false;">
                Búsqueda de empleados
                <input type="text" name="filtro" id="filtro">
                <input type="button" value="Filtrar"
onclick="filtrar()">
            </form>
        </div>
        <div id="listadoEmpleados">
            <c:import url="filtrarEmpleados.jsp">
                <c:param name="filtro" value=""/>
            </c:import>
        </div>
    </body>
</html>
```

filtrarEmpleados.jsp

```
 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>


        <jsp:useBean class="datos.Empresa" id="empresa"
scope="session"/>
        <table>
            <tr style="background-color: antiquewhite">
                <td>Nombre</td>
                <td>Apellidos</td>
                <td>Edad</td>
                <td>Departamento</td>
                <td></td>
            </tr>
            <c:forEach items="$
{empresa.filtrarEmpleados(param.filtro)}" var="empleado">
                <tr>
                    <td>${empleado.nombre}</td>
                    <td>${empleado.apellidos}</td>
                    <td style="text-align: right">$
{empleado.edad}</td>
                    <td>${empleado.departamento.nombre}</td>
                      <td>
                        <form action="editarEmpleado.jsp"
method="post">
                            <input type="hidden" name="id" value="$
{empleado.dni}">
                            <input type="submit" value="Editar">
                        </form>
                      </td>
                    </tr>
            </c:forEach>
        </table>
```