3-6-2016

Francisco Javier González Fernández | Joaquín Fernández León | Antonio Guzmán

GRUPO DE JAVI, JOAQUÍN Y ANTONIO

Índice

1. Diseño arquitectónico del sistema, vista de tipo funcional, vista física

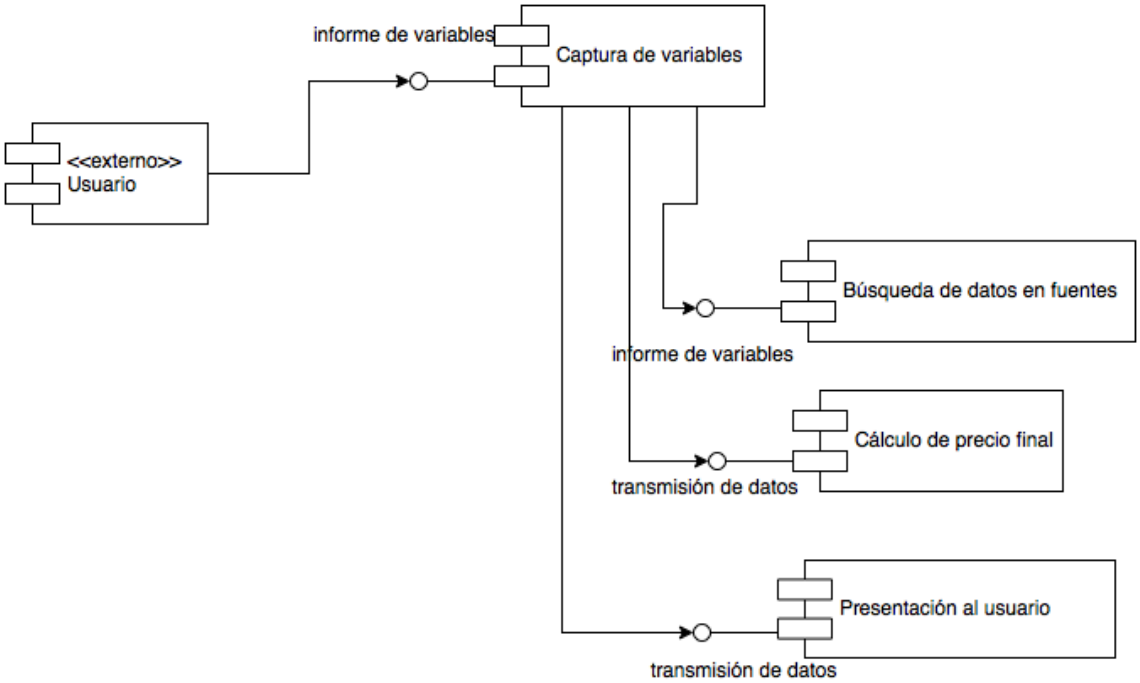
El diseño arquitectónico del sistema está compuesto por tres capas o módulos. El *módulo de interfaz de usuario, la capa de servicio* y el *módulo de acceso a los datos*.

En el *módulo de interfaz de usuario* o *capa de presentación* se ofrece un formulario con datos a rellenar por nuestro cliente. Se deben rellenar ciertos campos, para saber qué producto desea. Este campo pude ser el nombre del teléfono móvil, o directamente su código de producto. Además es posible recibir la región donde se desea recibir el envío del pedido.

La *capa de servicio* se encarga de buscar ofertas en las fuentes de información, añadir los valores adicionales, realizar comparaciones de precios y ordenarlas según el precio, o la tardanza de la entrega, en función de la elección del usuario.

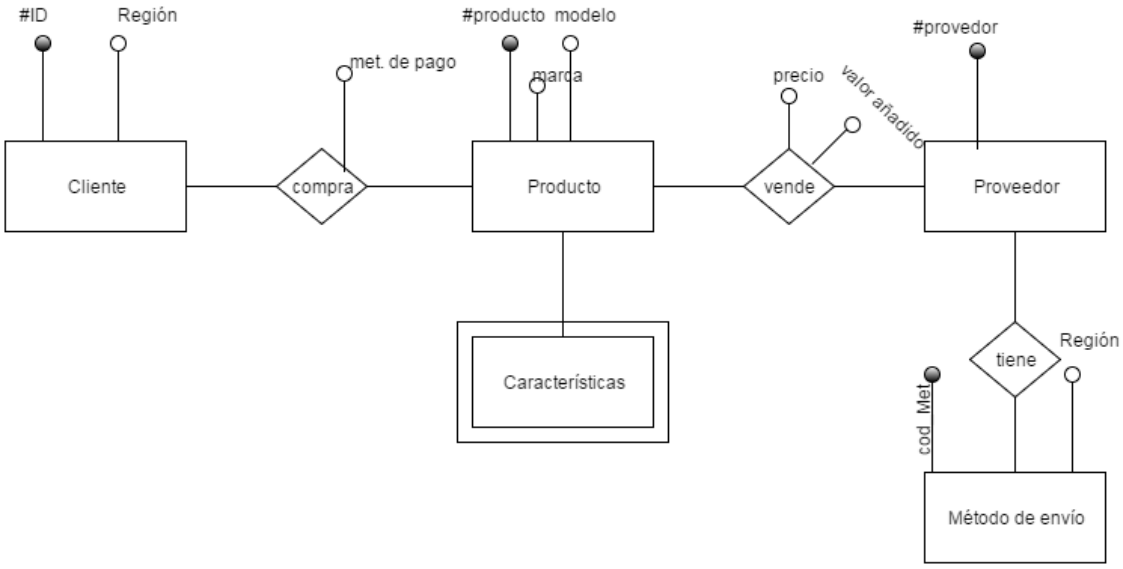
El *módulo de acceso a los datos* se encarga de acceder a la base de datos para consultar anteriores transacciones, integrar los datos y ofrecer a la capa superior datos necesarios para realizar cálculos necesarios.

Figura 1. Diseño del sistema desde el punto de vista funcional.



2. Modelo de datos del sistema

Figura 2. Diagrama entidad-relacción



Para manejar los datos se realizan los siguientes pasos. Primero, se guardan en la base de datos todas las consultas que ha hecho cada usuario. Es interesante hacer esto para posteriormente reutilizar los datos en próximas consultas y obtener un tiempo menor en las consultas, y para almacenar datos con la finalidad de obtener información y análisis sobre ellos.

La única información que es necesaria es la región de los clientes, es decir, desde dónde se realizará el pedido. Esta región del cliente se relaciona con la región de envío del proveedor y así obtener el valor añadido al precio del producto en función de los aranceles del lugar. Tendremos también un identificador que consistirá simplemente en claves autogeneradas.

Es necesario almacenar el método de pago. En función de él, puede conllevar a una serie de variaciones en el precio final.

Por otro lado hay que almacenar el precio de los productos. Este precio se almacena en la relación producto-proveedor porque los precios pueden variar con el tiempo. De esta manera, el precio final es calculado y puede variar sin necesidad de almacenarlo y, además, existe la posibilidad de informar al cliente de las variaciones del precio respecto a fechas pasadas.

Respecto al valor añadido de la relación *vende* es un valor calculado a partir de la región de envío del proveedor y la región del usuario. Sobre este valor influye además el método de envío elegido.

3. Diagrama de despliegue

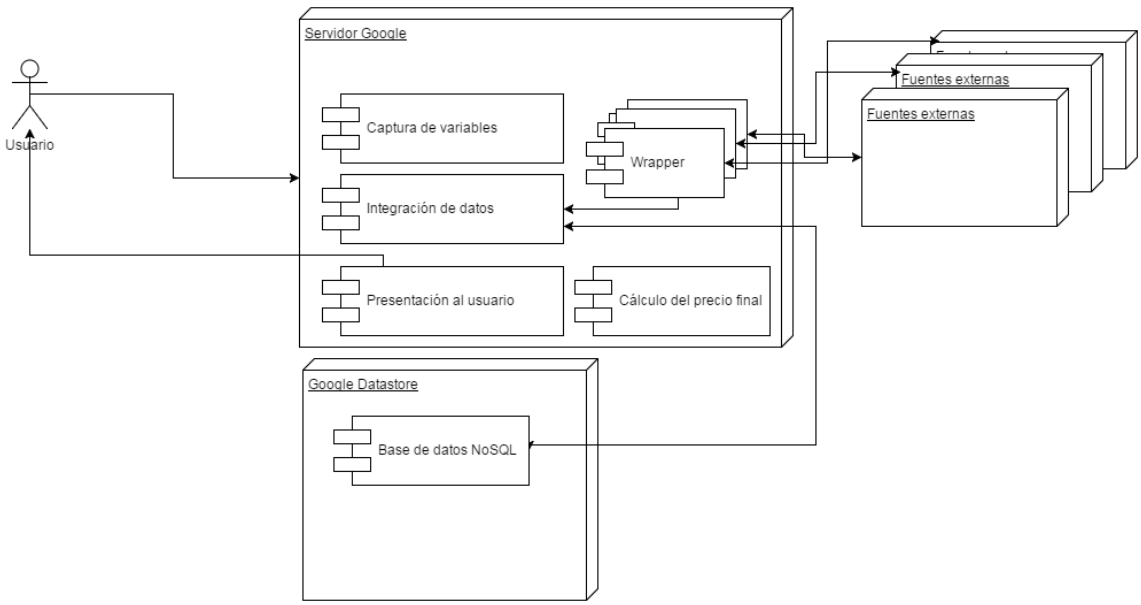
Con el diagrama de despliegue es posible modelar la disposición física de los artefactos software en nodos, componentes y asociaciones.

El usuario inicia la aplicación que interactúa con el servidor de Google y posteriormente le devuelve una respuesta al usuario.

El servidor de Google se encarga de recopilar todas las variables, calcular el precio final, acceder a la base de datos de Google Datastore y utilizar los wrappers para buscar en fuentes externas.

El Google Datastore contendrá la base de datos NoSQL.

Figura 3. Diagrama de despliegue



4. Hardware necesario para montar la aplicación en nuestra empresa.

PRODUCTO	MODELO	CARACTERÍSTICAS	UNIDADES	PRECIO (CON IVA)	SUMA
SWITCH	TP-LINK TL 561005D	5 puertos RJ45 Gigabit con detección automática de velocidad, Plug and Play	2	17,40 €	34,80 €
Router	Linksys ROUTER WRT - Router WiFi,	procesador de doble núcleo a 1,6 GHz, 1900 Mbps combinados: 1,3 Gbps (5 GHz) + 600 Mbps (2,4 GHz)	2	246,73 €	493,46 €
SERVIDOR WEB	WD myCloud EX4 | 8TB WD Purple	Puertos de suministro de alimentación y Ethernet dual, memoria de 512 MB y procesador 2.0 GHz	2	669,99 €	1.339,98 €
SERVIDOR DE DATOS	Synology Disk Station DS414	Servidor NAS - SATA 6Gb/s	2	399,00 €	798,00 €
				Total	2.666,24 €

5. Implementación

A continuación se explicará el funcionamiento de forma breve de los principales módulos que hacen posible el funcionamiento de la aplicación.

5.1 API de amazon

El funcionamiento consiste en hacer una petición con palabras clave introducidas por el usuario a través de la operación ItemSearch. Ahí introducimos la marca y el modelo del producto buscado, ordenando los resultados por precio.

A continuación, una vez obtenidos los ASIN de los productos, hacemos tantas peticiones como ASIN devueltos con la operación ItemLookup, indicándole el ASIN que buscamos, y de ahí, podemos obtener los detalles de cada producto que necesitamos mostrarle al cliente, como el asin, los atributos, el precio, una imagen, y el enlace de la oferta.

Una vez devueltos todos los detalles de los productos, los introducimos en un vector de objetos MobileInfo, explicaremos esto más detalladamente en el apartado del .jsp.

Para el establecimiento de parámetros en la búsqueda con la operación ItemSearch:

```
219             Map<String, String> params = new HashMap<String, String>();
220             params.put("Service", "AWSECommerceService");
221             params.put("Version", "2011-08-01");
222             params.put("AssociateTag", AMAZON_ASSOCIATE_TAG);
223             params.put("Operation", "ItemLookup");
224             params.put("Sort", "price");
225             params.put("ItemId", asin);
226             params.put("ResponseGroup", "OfferSummary");
227             String requestUrl1 = helper1.sign(params);
228             //String requestUrl2 = helper2.sign(params);
229             String queryString="&Version=2011-08-01" + "&AssociateTag=" + AMAZON_ASSOCIATE_TAG
230             + "&Operation=ItemLookup" +"&Sort="+"price"+ "&ItemId=" + asin +
231                 "&ResponseGroup="+"Medium,OfferSummary";
232             requestUrl1 = helper1.sign(queryString);
233             //requestUrl2 = helper2.sign(queryString);
234             System.out.println(requestUrl1);
235            // System.out.println(requestUrl2);
236         DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
237             DocumentBuilder db;
238         db = dbf.newDocumentBuilder();
239         Document doc1=null;
240         try {
241           doc1 = db.parse(requestUrl1);
242         } catch (IOException e) {
243           // TODO Auto-generated catch block
244           e.printStackTrace();
245         }
246         /*DocumentBuilderFactory dbf2 =DocumentBuilderFactory.newInstance();
```

Para recoger los resultados de esta búsqueda:

```
184         //Document doc2 = db2.parse(requestUrl2);
185         asin =getElementValue(doc1, "ASIN");
186
187
188
189             NodeList asinNodes = getElements(doc1,"ASIN");
190             String asinCodes[]=new String[asinNodes.getLength()];
191             for (int i=0; i<asinCodes.length; i++)
192               asinCodes[i] = asinNodes.item(i).getTextContent();
193
194
```

Guardamos los ASIN resultado en un vector, que después usaremos para hacer la búsqueda con la operación ItemLookup.

```
219             Map<String, String> params = new HashMap<String, String>();
220             params.put("Service", "AWSECommerceService");
221             params.put("Version", "2011-08-01");
222             params.put("AssociateTag", AMAZON_ASSOCIATE_TAG);
223             params.put("Operation", "ItemLookup");
224             params.put("Sort", "price");
225             params.put("ItemId", asin);
226             params.put("ResponseGroup", "OfferSummary");
227             String requestUrl1 = helper1.sign(params);
228             //String requestUrl2 = helper2.sign(params);
229             String queryString="&Version=2011-08-01" + "&AssociateTag=" + AMAZON_ASSOCIATE_TAG
230             + "&Operation=ItemLookup" +"&Sort="+"price"+ "&ItemId=" + asin +
231                 "&ResponseGroup="+"Medium,OfferSummary";
232             requestUrl1 = helper1.sign(queryString);
233             //requestUrl2 = helper2.sign(queryString);
234             System.out.println(requestUrl1);
235            // System.out.println(requestUrl2);
236         DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
237             DocumentBuilder db;
238         db = dbf.newDocumentBuilder();
```

Después de establecer los parámetros para hacer la petición de ItemLookup:

```
259         //busqueda Precios
260         NodeList LowestNodes = getElements(doc1,"LowestNewPrice");
261             String LowestPrices[]=new String[LowestNodes.getLength()];
262             String prices[]=new String [LowestPrices.length];
263 
264             for (int i=0; i<LowestPrices.length; i++){
265               LowestPrices[i] = LowestNodes.item(i).getTextContent();
266               prices[i]="";
267 
268 
269               for(int j=LowestPrices[i].indexOf("$");j<LowestPrices[i].length();j++){
270                 prices[i]+=LowestPrices[i].charAt(j);
271               }
272 
273             }
```

Empezamos a recoger los resultados que nos interesan, y guardándolos en vectores que después usaremos para crear los objetos que debemos pasarle al .jsp.

```
274            //Busqueda imagenes
275            NodeList ImageNodes = getElements(doc1,"SmallImage");
276            String Images[]=new String[ImageNodes.getLength()];
277            String images_final[]=new String [Images.length];
278
279            for (int i=0; i<Images.length; i++){
280              Images[i] = ImageNodes.item(i).getTextContent();
281              images_final[i]="";
282
283              boolean acabado=false;
284              for(int j=Images[i].indexOf("h");j<Images[i].length() &&!acabado;j++){
285                if(j>3){
286                  if(Images[i].charAt(j-3)=='.'&&Images[i].charAt(j-2)=='j'&&Images[i].charAt(j-1)=='p'&&Images[i].charAt(j)=='g'){
287                    acabado=true;
288                  }
289                }
290                images_final[i]+=Images[i].charAt(j);
291              }
292
293              System.out.println("Esta es la imagen: "+images_final[i]);
294
295
296            }
```

Aquí estamos obteniendo las imágenes, guardándolo en su vector correspondiente.

```
297              //busca modelos
298            NodeList titleNodes =getElements(doc1, "Title");
299            String titles[]=new String[titleNodes.getLength()];
300             for(int i=0;i<titles.length;i++){
301               titles[i]=getElementValue(doc1, "Title");
302             }
303             NodeList brandNodes =getElements(doc1, "Brand");
304              String brand[]=new String[brandNodes.getLength()];
305               for(int i=0;i<brand.length;i++){
306                 brand[i]=getElementValue(doc1, "Brand");
307               }
308            NodeList linkNodes =getElements(doc1, "DetailPageURL");
309                String link[]=new String[linkNodes.getLength()];
310                 for(int i=0;i<link.length;i++){
311                   link[i]=getElementValue(doc1, "DetailPageURL");
312             }
313
314
315                for(int i=0;i<prices.length;i++){
316                  MobileInfo nuevo= new MobileInfo(resultados[i], titles[i], prices[i], brand[i],images_final[i],link[i]);
317                  mobiles.add(nuevo);
318                }
319
```

Por último, obtenemos la marca, los detalles, y los enlaces de cada producto.

Creamos todos los objetos Mobile, los metemos en nuestro vector, y más adelante se lo pasaremos al .jsp para que muestre los resultados.

5.2. Web Scrapper

Para la construcción del web scrapper, nos hará falta un lector de html, el cual hemos desarrollado como una clase auxiliar:

```java

public class LectorHTML {

  public static void main(String[] args) {
    // TODO Auto-generated method stub
  }

  public String leeHtml(String url){
    URL urlObjet;
    String codigo = "";
    String linea;
    try{
      URL urlObject=new URL(url);
      InputStreamReader isr=new InputStreamReader(urlObject.openStream());
      BufferedReader br = new BufferedReader(isr);
      while((linea=br.readLine())!=null){
        codigo+=linea;
      }

    }catch (MalformedURLException e){
      e.printStackTrace();
    }catch (IOException e){
      e.printStackTrace();
    }
    return codigo;
  }

}
```

De esta manera, dada una url, el método leeHTML nos devolverá el código .html de la página, y de ahí extraeremos los datos que necesitamos.

Como vemos en la siguiente figura, construimos el enlace con las palabras clave que nos da el usuario, que proviene del index, y en ese html, de los resultados de esa búsqueda, nos quedamos con el enlace de los detalles de la oferta, y de ahí, sacamos el precio y las características.

```
39      public String buscaEnPcComponentes(String codigoProducto){
40      LectorHTML lector = new LectorHTML();
41      String enlaceParte1, enlaceParte2, resultado= new String();
42      enlaceParte2 = "?p="+ parteCadena(codigoProducto);
43      enlaceParte1 = "http://www.pccomponentes.com/buscar.php";
44      resultado = lector.leeHtml(enlaceParte1 + enlaceParte2);
45
46      int claveBusqueda = resultado.indexOf("data-href");
47      if (claveBusqueda < 0){
48        return "Error";
49      }
50      claveBusqueda+=11; // Saltamos la cadena data-href="
51      String provisional, datosProducto = new String();
52      provisional="";
53      char letra = 'a';
54      while(letra != '"'){
55        letra = resultado.charAt(claveBusqueda);
56        if(letra!='"'){
57          provisional+=letra;
58        }
59        claveBusqueda++;
60      }
61
62      enlacePC = provisional;
63      resultado = lector.leeHtml(provisional);
64
65      claveBusqueda = resultado.indexOf("itemprop=\"name");
66      claveBusqueda +=16; //saltamos la cadena itemprop="name">
67      provisional="";
68      letra='a';
69      while(letra != '<'){
70        letra = resultado.charAt(claveBusqueda);
71        if(letra!='<'){
72          provisional+=letra;
73        }
74        claveBusqueda++;
```

```
76      caracteristicasPC = provisional;
77
78
79      Float precio;
80      claveBusqueda = resultado.indexOf("data-price");
81      claveBusqueda+=12; // Saltamos la cadena data-price="
82      provisional="";
83      letra = 'a';
84      while(letra != '"'){
85        letra = resultado.charAt(claveBusqueda);
86        if(letra!='"'){
87          provisional+=letra;
88        }
89        claveBusqueda++;
90      }
91      String precioFinal = new String();
92      precioFinal = "";
93      for(int i=0; i<provisional.length(); i++){
94        if (provisional.charAt(i)==','){
95          precioFinal+='.';
96        }else
97          precioFinal+=provisional.charAt(i);
98
99      }
100
101     return precioFinal;
102   }
```

5.3 Muestra de resultados a través del jsp

Primero, tenemos que enviar al jsp la estructura de datos que queramos, en nuestro caso, un Array de objetos móvil, el cual recorrerá el jsp y mostrará uno a uno.

Eso lo hacemos mediante la sentencia de la línea 382-383.

```
364   public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
365     String marca = req.getParameter("marca");
366     String modelo = req.getParameter("modelo");
367     StringBuilder formatModelo=new StringBuilder();
368     String modeloFormateado;
369     modeloFormateado=parteCadena(modelo);
370     int i=0;
371     resp.getWriter().println("El precio del producto en PCComponentes es de: " + buscaEnPcComponentes(modeloFormateado));
372     resp.getWriter().println("enlace de la oferta: " + enlacePC);
373     resp.getWriter().println("Para el producto: " + caracteristicasPC);
374     System.out.println(modeloFormateado);
375
376
377     buscarEnAmazon(req, resp, marca, modeloFormateado);
378     for(int j=0; j<resultados.length;j++)
379       extraerInformacion(resultados[j]);
380       MobileInfo nuevo=new MobileInfo(null, caracteristicasPC,buscaEnPcComponentes(modeloFormateado),marca,null,enlacePC);
381       mobiles.add(nuevo);
382      req.getSession().setAttribute("moviles", mobiles);
383      resp.sendRedirect("http://localhost:8888/MobileInfo.jsp");
384
385   }
386
387
388 }
```

Así, le enviaremos al jsp el Array mencionado.

```
<%
    ArrayList<MobileInfo> moviles = (ArrayList<MobileInfo>) session.getAttribute("moviles");
  MobileInfo movil;
  %>
  <h1>Tu comprarador de precios de m?viles</h1>
  <br/>

  <h3>El resultado de su b?squeda es el siguiente:</h3>
  <br/>
  <%
  for(int i=0; i<moviles.size(); i++){
    movil = moviles.get(i);

    if(movil != null && !movil.equals(null)){

  %>
```

Ahora, en el jsp, estamos recogiendo el dato que nos han enviado a través de una variable de sesión, y vamos a recorrerlo y a mostrarlo.

```
    <form>
      <table>
        <tr>
        <img src="<%out.println(movil.getImage());%>" alt="No se puede mostrar la imagen"  height="300" width="220" style="float:left; margin:10px;" >
        </tr>
          <tr>
              <td><input type="text" value="ASIN: <%out.println(movil.getAsin());%>" readonly="readonly"></td>
          </tr>
          <tr>
          <td><input type="text" value="Marca: <%out.println(movil.getMarca());%>" readonly="readonly" id="marca"></td>
        </tr>
        <tr>
            <td><input type="text" value="Modelo: <%out.println(movil.getModelo());%>" readonly="readonly"></td>
          </tr>
          <tr>
            <td><input type="text" value="Precio: <%out.println(movil.getPrecio());%>" readonly="readonly"></td>
          </tr>
          <tr>
            <td><a href="<%out.println(movil.getEnlace());%>"><p>Ver oferta</p></a></td>
          </tr>


      </table>
      <br>
    </form>
  <%} %>
<%} %>
```

Mostramos los resultados.