Comercio electrónico con AJAX

Web

Introducción

A esta altura todos conocen qué es un sitio web de comercio electrónico. El asunto es, ¿qué puede aportar una aplicación rica de Internet en un desarrollo de este tipo? Aportará muchas características interesantes, que harán aumentar la cantidad de compras que se hagan en nuestro sitio web. Un sitio web rico tiene mayor tasa de conversión; esto es, la gente compra más cuando se trata de una aplicación rica.

A continuación se detallan algunas características de un comercio electrónico con AJAX:

- El proceso de compra es notoriamente más veloz.
- Es posible obtener estadísticas detalladas del comportamiento del usuario para ofrecerle sugerencias, ofertas o paquetes de productos.
- El usuario puede interactuar mejor con el sistema, puede comparar productos, arrastrar productos al canasto de compras o tener su lista de productos deseados (wish list).
- El usuario nunca sale de una sola interfaz donde navega por categorías, busca, elije y compra.

Estructura

Lenguaje de servidor

En los ejemplos anteriores del libro se habían elegido PHP y MySQL como solución del lado del servidor. Para demostrar que AJAX en realidad puede comunicarse con cual-

quier plataforma del servidor, se va a hacer uso de la tecnología ASP.NET con lenguaje Visual Basic y con SQL Server. Fácilmente se podría haber utilizado también C#, IronPython u otro lenguaje .NET.

Para los que no conozcan la tecnología de ASP.NET, pueden descargarse en forma gratuita el editor Visual Web Developer Express y el motor de base de datos SQL Server Express. Asimismo, este proyecto será fácil de migrar a PHP u otra plataforma.

Base de datos

Se va a utilizar SQL Server como motor de base de datos con la estructura de un e-commerce que se puede ver en la figura 1. La estructura es simple, y lo es a los efectos de demostrar las capacidades de AJAX en un sitio de estas características.

Un sitio en producción debería tener mayores capacidades, cálculo de estadísticas y probablemente mayor información y filtros acerca de los productos. También se debería almacenar más información del usuario que hace la compra.

Nótese que en la tabla pedidos hay un campo finalizado. La idea es ofrecer un servicio de autoguardado que guarde el pedido en el servidor por si se cierra el navegador o el usuario tiene algún problema. Esto es un servicio muy útil en aplicaciones ricas de Internet, que le ofrece al usuario una buena experiencia de navegación.

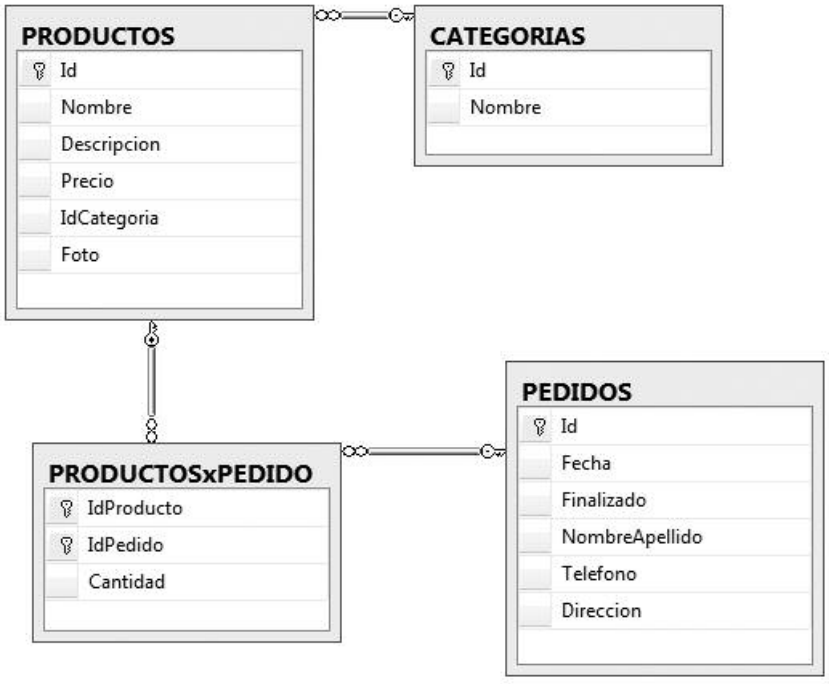

Fig. 1. Diagrama de la base de correo electrtónico.

Clases de conexión

Así como en PHP había una clase para conectarse a la base de datos y centralizar así el acceso, se hará lo mismo en .NET. El string de conexión a la base de datos, que incluye usuario y password, también podría incorporarse en el archivo de configuración web.config.

Esta clase es BD.vb y sólo se la debe ubicar en la carpeta App_Code del proyecto web.

```
Imports Microsoft.VisualBasic
Imports System.Data
Imports System.Data.SqlClient

Public Class BD

    Public Shared _conexion As SqlConnection

    Public Shared Property Conexion() As SqlConnection
        Get
            If _conexion Is Nothing Then
                _conexion = New SqlConnection("Data
Source=nombre_server;Initial Catalog=ecommerce;Persist Security
Info=True;User ID=usuario;Password=pwd")
            End If
            Return _conexion
        End Get
        Set(ByVal value As SqlConnection)
            _conexion = value
        End Set
    End Property

    Public Sub New()
    End Sub

    Public Shared Sub Abrir()
        If Conexion.State <> ConnectionState.Open Then
            Conexion.Open()
        End If
    End Sub


    Public Shared Sub Cerrar()
        If Conexion.State <> ConnectionState.Closed Then
            Conexion.Close()
        End If
    End Sub

    Public Shared Sub ExecuteNonQuery(ByVal cmd As String)
        Abrir()
        Dim com As New SqlCommand(cmd, Conexion)
```

```
        com.ExecuteNonQuery()
        Cerrar()
    End Sub

    Public Shared Sub ExecuteNonQuery(ByVal cmd As SqlCommand)
        Abrir()
        cmd.Connection = Conexion
        cmd.ExecuteNonQuery()
        Cerrar()
    End Sub

    Public Shared Function ExecuteScalar(ByVal cmd As String) As Object
        Abrir()
        Dim com As New SqlCommand(cmd, Conexion)
        Return com.ExecuteScalar()
        Cerrar()
    End Function

    Public Shared Function ExecuteScalar(ByVal cmd As SqlCommand) As
Object
        Abrir()
        cmd.Connection = Conexion
        Return cmd.ExecuteScalar()
        Cerrar()
    End Function

    Public Shared Function Execute(ByVal cmd As String) As DataTable
        Dim comando As New SqlCommand(cmd, Conexion)
        comando.Connection = Conexion
        Dim adap As New SqlDataAdapter(comando)
        Dim dt As New DataTable
        adap.Fill(dt)
        Return dt
    End Function

    Public Shared Function Execute(ByVal cmd As SqlCommand) As DataTa-
ble
        cmd.Connection = Conexion
        Dim adap As New SqlDataAdapter(cmd)
        Dim dt As New DataTable
        adap.Fill(dt)
        Return dt
    End Function



End Class
```

Con esta clase se tendrá centralizado el acceso para realizar consultas a la base de datos en forma sencilla. Se ofrece una solución similar a la observada antes en PHP.

Diseño web

El diseño del sitio de comercio electrónico se basará en una Full Rich Internet Application, o sea, una sola URL donde el usuario podrá realizar todas las acciones de la aplicación web.

La página web tendrá las zonas siguientes (fig. 2):

- **Zona de menú:** permitirá acceder a distintas opciones de menú, como buscar por categorías, buscar por palabra o ver el canasto de compras.
- **Zona de resultados:** mostrará el resultado de una búsqueda con el nombre y el precio del producto.
- **Zona de detalle:** mostrará el detalle de un producto seleccionado. Esta información puede ubicarse debajo de cada resultado o en una zona predefinida.
- **Zona de canasto:** permitirá ver todos los productos que hay en el canasto; se pueden modificar cantidades, eliminar productos y limpiar el canasto.
- **Zona de finalización de compra:** confirmará los datos de envío de los productos y finalizará la compra.

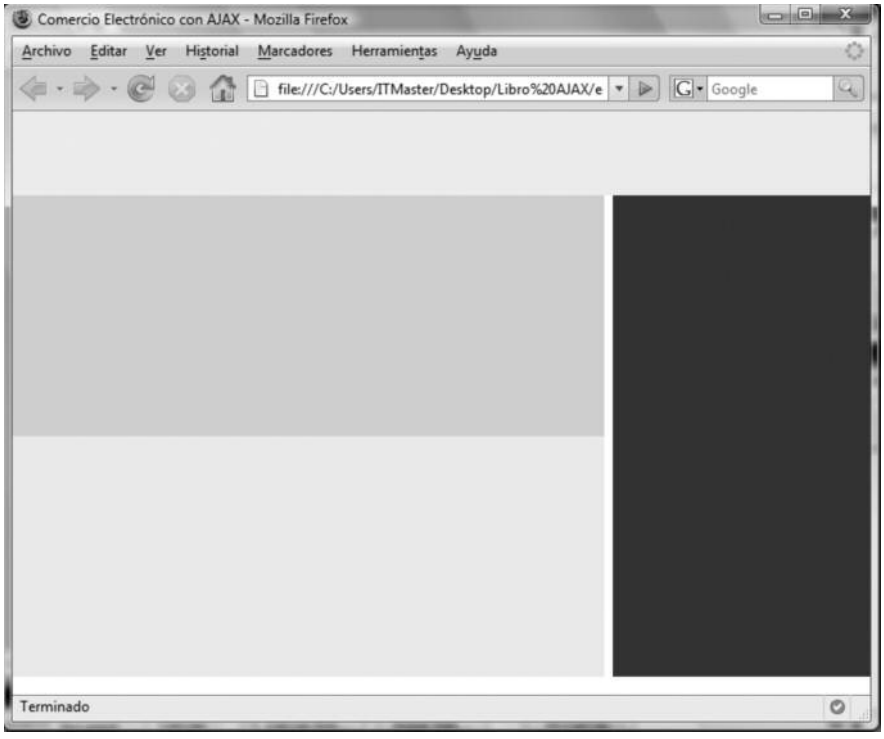

Fig. 2. Esqueleto del diseño de comercio electrónico con XHTML y CSS. Está realizado con cajas de colores para identificar mejor las zonas en los ejemplos.

Se tendrá la estructura XHTML siguiente:

```
<!DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Transitional//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head>
	<meta http-equiv=”Content-Type” content=”text/html;
charset=iso-8859-1” />
	<title>Comercio Electrónico con AJAX</title>
	<link href=”estilos.css” type=”text/css” rel=”stylesheet” />
</head>

<body>

	<div id=”menu”></div>

	<div id=”contenedor”>
		<div id=”canasto”></div>
		<div id=”resultados”></div>
		<div id=”detalle”></div>
	</div>

	<div id=”finalizacion”></div>

</body>
</html>
```

Y el CSS adjunto

```
* {
	font-family:Verdana, Arial, Helvetica, sans-serif;
	font-size: 11;
	padding: 0px;
	margin: 0px;
}


#menu {
	width: 100%;
	background-color:#CCFF99;
	height: 70px;
}


#resultados {
	width: 69%;
	background-color: #FFCCCC;
	height: 250px;
	float: left;
}
```

```
#detalle {
	width: 69%;
	background-color: #EEEEEE;
	height: 250px;
	float: left;
}

#canasto {
	float: right;
	width: 30%;
	background-color:#0033CC;
	color: white;
	height: 500px;
}
```

Se va a ir armando el menú de navegación (fig. 3):

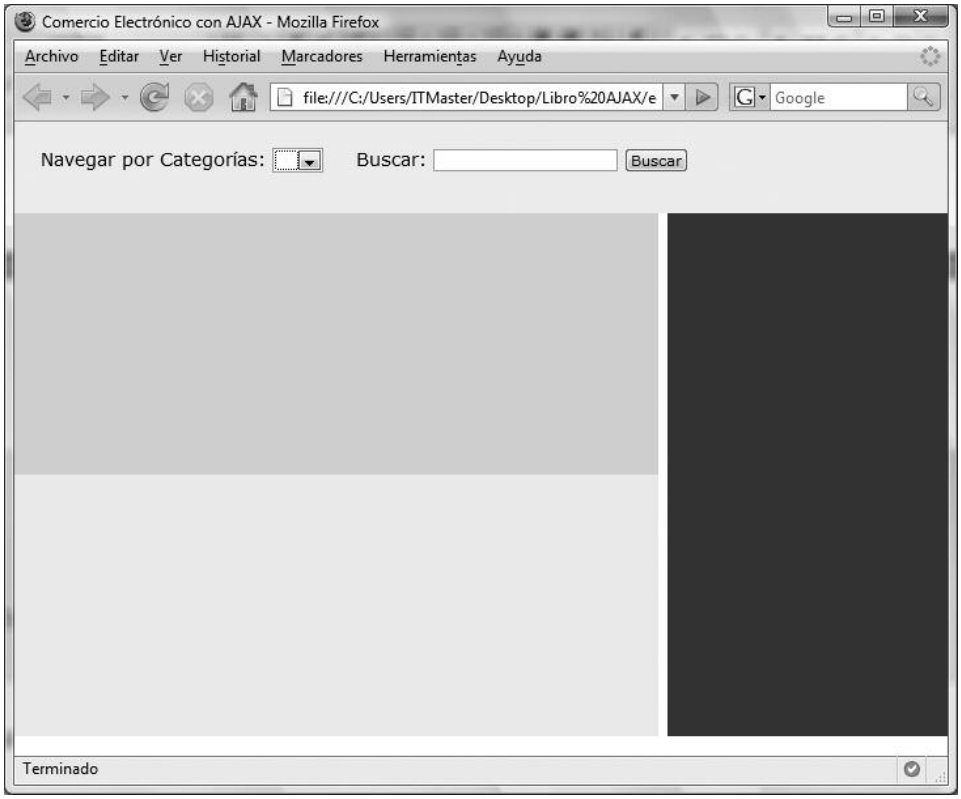

Fig. 3. Se agregan opciones al menú de navegación.

```
<div id="menu">
            <div id="menuOpciones">
                  Navegar por Categorías:
                  <select id="listaCategorias">
                  </select>

                  Buscar:
                  <input type="text" id="textoBusqueda" />
                  <input type="button" id="btnBuscar" value="Buscar" />
            </div>
      </div>
```

Y el siguiente CSS:

```
#menuOpciones {
      height: 70px;
      vertical-align: middle;
      padding: 20px;
      font-size: larger;
}

#listaCategorias {
      margin-right: 20px;
}
```

Comportamiento inicial

Ahora también se le agrega algo de información a cada sección y se van incorporando los archivos de inclusión de JavaScript. Los JavaScript que importaremos son:

- **AjaxLib.js**: nuestra librería de trabajo AJAX.
- **index.js**: el controlador de nuestra aplicación.
- **Prototype.js**: la librería Prototype vista con anterioridad en el libro.
- **Scriptaculous.js**: la librería Script.aculo.us. Recuérdese que este JS permitirá cargar el resto de los .js necesarios.

También se agregará un DIV que permitirá mostrar un aviso de cargando. Con todos estos cambios el código queda así (fig. 4):

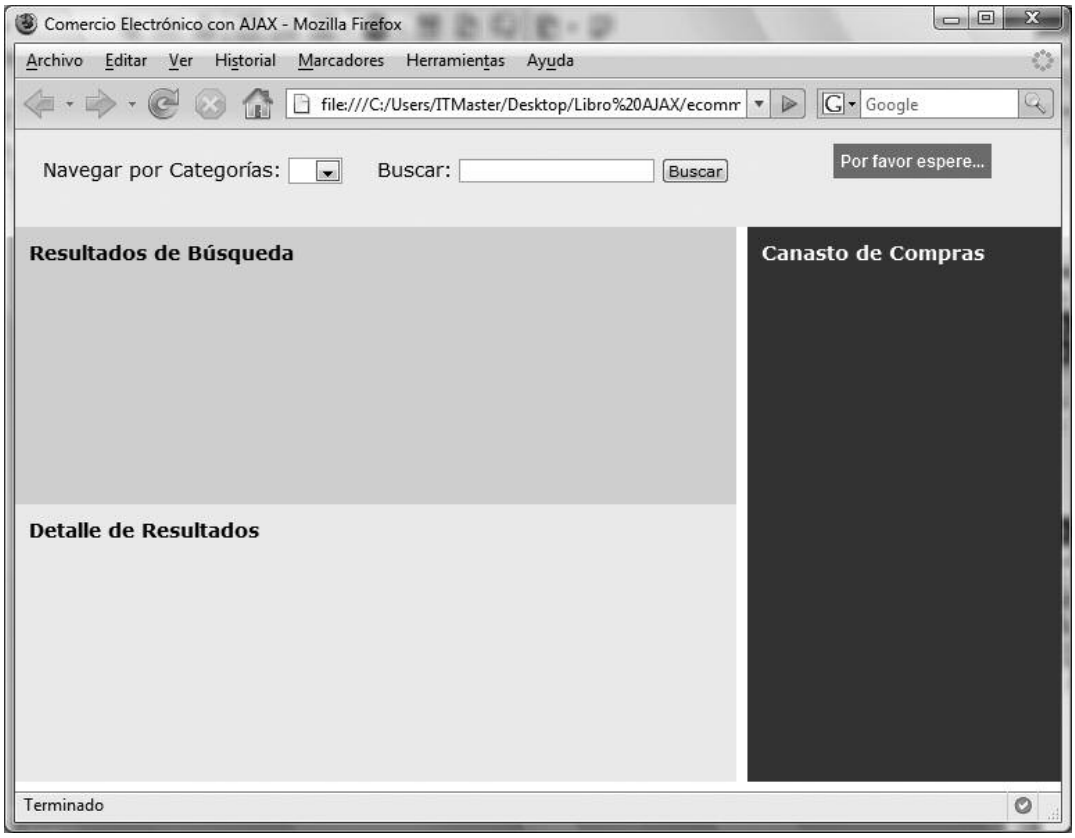

Fig. 4. Visualmente vamos encontrando la ubicación de cada uno de los elementos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
      <title>Comercio Electrónico con AJAX</title>
      <link href="estilos.css" type="text/css" rel="stylesheet" />
      <script type="text/javascript" src="prototype.js"></script>
      <script type="text/javascript" src="scriptaculous.js"></script>
      <script type="text/javascript" src="AjaxLib.js"></script>
      <script type="text/javascript" src="index.js"></script>
</head>

<body>

      <div id="menu">
            <div id="menuOpciones">
                  Navegar por Categorías:
                  <select id="listaCategorias">
                  </select>
```

```
                    Navegar por Categorías:
                    <select id="listaCategorias">
                    </select>

                    Buscar:
                    <input type="text" id="textoBusqueda" />
                    <input type="button" id="btnBuscar" value="Buscar" />
              </div>
        </div>

        <div id="contenedor">
              <div id="canasto">
                    <h2>Canasto de Compras</h2>
                    <div id="listaCanasto">
                    </div>
              </div>
              <div id="resultados">
                    <h2>Resultados de Búsqueda</h2>
                    <div id="listaResultados">
                    </div>
              </div>
              <div id="detalle">
                    <h2>Detalle de Resultados</h2>
                    <div id="listaDetalle">
                    </div>
              </div>
        </div>

        <div id="cargando">Por favor espere...</div>

        <div id="finalizacion"></div>

</body>
</html>
```

Y el archivo de estilos quedaría como el siguiente:

```
*
{
        font-family:Verdana, Arial, Helvetica, sans-serif;
        font-size: 11px;
        padding: 0px;
        margin: 0px;
}

#menu
{
        width: 100%;
        background-color:#CCFF99;
        height: 70px;
}

#menuOpciones
{
```

```
        height: 70px;
        vertical-align: middle;
        padding: 20px;
        font-size: larger;
}

#listaCategorias
{
        margin-right: 20px;
}

#contenedor h2
{
        padding: 10px;
        font-size: larger;
}

#resultados
{
        width: 69%;
        background-color: #FFCCCC;
        height: 200px;
        float: left;
}

#detalle
{
        width: 69%;
        background-color: #EEEEEE;
        height: 200px;
        float: left;
}

#canasto
{
        float: right;
        width: 30%;
        background-color:#0033CC;
        color: white;
        height: 400px;
}

#cargando
{
        position: absolute;
        top: 10px;
        right: 50px;
        background-color: red;
        color: white;
        font-family: Arial, Helvetica, sans-serif;
        font-size: 12px;
        font-weight: bold;
        padding: 5px;
}
```

Ahora se verán los pasos a seguir en el controlador:

1. Traer vía AJAX todas las categorías para completar la lista.
2. Cargar los productos disponibles en la primera categoría.
3. Darle funcionalidad a la lista de categorías y al buscador.
4. Apagar el cartel de cargando.

Luego, se irá incorporando el resto de las funciones por medio de JavaScript y CSS (fig. 5).

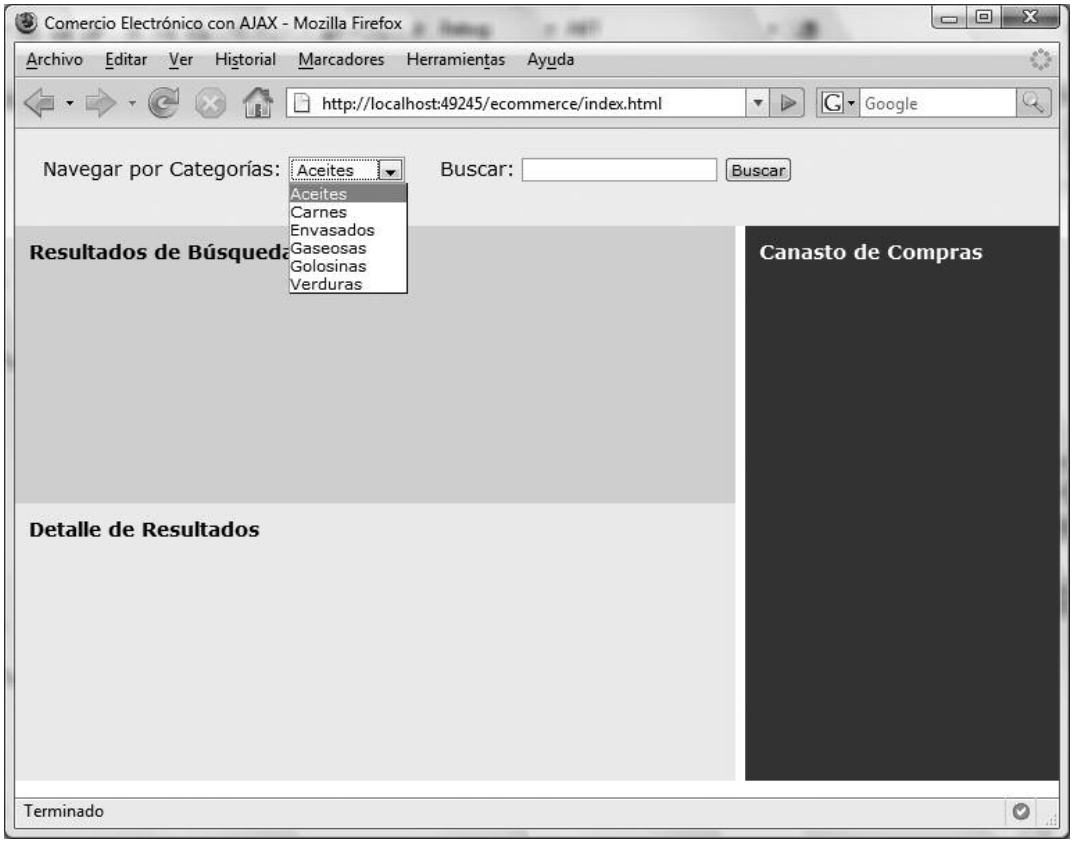

Fig. 5. Vía AJAX ya se cuenta con el listado de Categorías funcionando.

Index.js

```
window.onload = function() {
    $Ajax("categorias.aspx", {
        onfinish: cargarCategorias,
        tipoRespuesta: $tipo.JSON,
        onerror: function(e) { alert(e) }
    });

    $("btnBuscar").onclick = enviarBusqueda;
    $("listaCategorias").onchange = buscarCategoria;
}


function cargarCategorias(lista) {
    // Itero entre cada categoría recibida
    for (var i=0; i<lista.length; i++) {
        var cat = lista[i];
        // Agrego la categoría a la lista de Opciones
        var opc = new Option(cat.nombre, cat.id);
        $("listaCategorias").options[$("listaCategorias").length] = opc;
    };
    // Apago el cartel de Loading
    $("cargando").hide();
}

function enviarBusqueda() {
    $Ajax("buscar.aspx", {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON
    });
}

function buscarCategoria() {
    $Ajax("categoria.aspx?id=" + $F("listaCategorias"), {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON
    });
}

function cargarResultados() {
    // TODO
}
```

Resta ver el código de categorías.aspx en Visual Basic. Recuérdese que se podría haber utilizado cualquier otro lenguaje .NET, como C# o incluso compilar esta funcionalidad en clases separadas.

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<script runat="server">

    " ' <summary>
    " ' Lee las categorías y las devuelve en formato JSON
    " ' </summary>
    " ' <remarks></remarks>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
                                System.EventArgs)
        'Este evento se ejecuta cuando se carga la página en el
        servidor
        Dim Resultado As DataTable
        Resultado = BD.Execute("SELECT * FROM CATEGORIAS ORDER BY
                                NOMBRE")

        Dim Json As String = "["

        For Each Registro As DataRow In Resultado.Rows
            'Voy armando el JSON de cada categoría
            Json += "{"
            Json += "id: " + Registro("id").ToString + ", "
            Json += "nombre: '" + Registro("nombre") + " '"
            Json += "}"

            'Coma que separa cada objeto con el siguiente
            Json += ","
        Next

        'Hay que eliminar la última coma
        Json = Json.Remove(Json.Length - 1, 1)

        Json += "]"

        Response.Write(Json)
    End Sub
</script>
```

Resultados de búsqueda

Teniendo en cuenta que nuestro sistema puede tener muchos productos incorporados, desde el servidor se van a enviar el id, el nombre y el precio de los productos a incorporar en el canasto en los resultados de búsqueda.

Recuérdese que se pueden recibir resultados de dos maneras:

- Por selección de categoría.
- Por búsqueda por palabra.

Los dos scripts de servidor que devolverán los resultados son los siguientes:

categoria.aspx

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">

    " ' <summary>
    " ' Busca Productos y los devuelve en formato JSON
    " ' </summary>
    " ' <remarks></remarks>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As Sys-
tem.EventArgs)
        'Este evento se ejecuta cuando se carga la página en el
        servidor
        Dim Resultado As DataTable

        'Leo el ID desde un parámetro GET
        'Convierto a Entero el ID para evitar problemas de seguridad
        Dim id As Integer
        Try
            id = Integer.Parse(Request.QueryString("id"))
        Catch ex As Exception
            id = 0
        End Try
        If id > 0 Then
            Dim sql As String
            sql = "SELECT * FROM PRODUCTOS WHERE IdCategoria = " +
                            id.ToString
            Resultado = BD.Execute(sql)

            Dim Json As String = "["

            For Each Registro As DataRow In Resultado.Rows
                'Voy armando el JSON de cada producto
                Json += "{"
                Json += "id: " + Registro("id").ToString + ", "
                Json += "nombre: '" + Registro("nombre") + "', "
                Json += "precio: " + Registro("precio").ToString
                Json += "}"

                'Coma que separa cada objeto con el siguiente
                Json += ","
            Next

            'Hay que eliminar la última coma
            Json = Json.Remove(Json.Length - 1, 1)
```

```
            Json += "]"

            Response.Write(Json)

        End If
    End Sub
</script>
```

buscar.aspx (fig. 8-6)

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">

    " ' <summary>
    " ' Busca Productos y los devuelve en formato JSON
    " ' </summary>
    " ' <remarks></remarks>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
                                System.EventArgs)
        'Este evento se ejecuta cuando se carga la página en el
        servidor
        Dim Resultado As DataTable

        'Lee las palabras a buscar desde un parámetro GET
        Dim palabra As String = Request.QueryString("buscar")
        If palabra IsNot Nothing Then
            'Eliminamos comillas para evitar problemas de seguridad
            palabra = palabra.Replace(" '", "")
            Dim sql As String
            sql = "SELECT * FROM PRODUCTOS WHERE NOMBRE LIKE '%" + _
                    palabra + "% ' OR DESCRIPCION LIKE '%" + palabra
                    + "% '"
            Resultado = BD.Execute(sql)

            Dim Json As String = "["

            For Each Registro As DataRow In Resultado.Rows
                'Voy armando el JSON de cada producto
                Json += "{"
                Json += "id: " + Registro("id").ToString + ", "
                Json += "nombre: '" + Registro("nombre") + " ', "
                Json += "precio: " + Registro("precio").ToString
                Json += "}"

                'Coma que separa cada objeto con el siguiente
```

```
                Json += ","
            Next

            'Hay que eliminar la última coma
            Json = Json.Remove(Json.Length - 1, 1)

            Json += "]"

            Response.Write(Json)

        End If
    End Sub
</script>
```

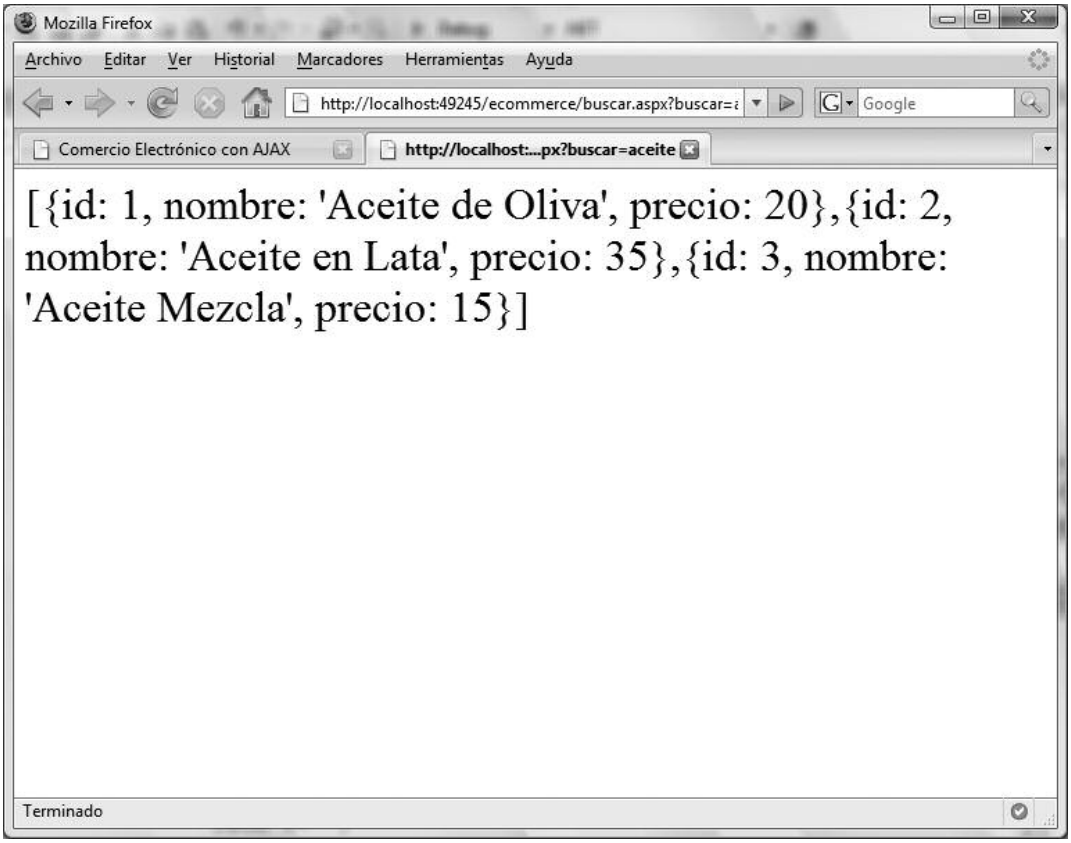

Fig. 6. Se observa una respuesta del buscar.aspx en formato JSON.

Ahora queda terminar el JavaScript para que muestre los resultados de búsqueda.

```
window.onload = function() {
    $Ajax("categorias.aspx", {
        onfinish: cargarCategorias,
        tipoRespuesta: $tipo.JSON,
        onerror: function(e) { alert(e) }
    });

    $("btnBuscar").onclick = enviarBusqueda;
    $("listaCategorias").onchange = buscarCategoria;
}


function cargarCategorias(lista) {
    // Itero entre cada categoría recibida
    for (var i=0; i<lista.length; i++) {
        var cat = lista[i];
        // Agrego la categoría a la lista de Opciones
        var opc = new Option(cat.nombre, cat.id);
        $("listaCategorias").options[$("listaCategorias").length] = opc;
    };

    // Cargo los productos de la primera categoría
    buscarCategoria();
}

function enviarBusqueda() {
    $Ajax("buscar.aspx?buscar=" + $F("textoBusqueda"), {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON,
        divCargando: "cargando"
    });
}

function buscarCategoria() {
    $Ajax("categoria.aspx?id=" + $F("listaCategorias"), {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON,
        divCargando: "cargando"
    });
}


var listaResultados;

function cargarResultados(lista) {
    // Apago el cartel de Loading
    $("cargando").hide();

    // Guardo la lista en una variable global
    // para posible uso futuro
    listaResultados = lista;

    // Limpio la lista de resultados vieja
    $("listaResultados").innerHTML = "";
```

```
    // Itero entre los resultados
    for (var i=0; i<lista.length; i++) {
        agregarResultado(lista[i]);
    }
}

function agregarResultado(producto) {
    var div = "<div class='productoResultado' " +
              "id='producto" + producto.id + " '>";
    div += "<div class='nombreProducto' onclick='detalle(" + producto.id
             + ")' >" + producto.nombre + "</div>";
    div += "<div class='precioProducto'>$ " + producto.precio +
                  "</div>";
    div += "<div class='agregarProducto' onclick='agregar(" +
                  producto.id
             + ")'>Agregar</div>";
    div += "</div>"
    $("listaResultados").innerHTML += div;    }
```

Nótese que se agregan en forma dinámica unos DIV con cada fila de resultado de un producto. Para ello se utilizan algunos estilos CSS nuevos como los siguientes (figs. 8-7 y 8-8):

```
#listaCanasto, #listaResultados, #listaDetalle
{
       padding: 15px;
       overflow: auto;
       height: 60%;
}


.productoResultado
{
       width: 470px;
       padding: 4px;
       margin: 2px;
       background-color: White;
       font-size: large;
       height: 15px;
}


.productoResultado:hover
{
       background-color: #EEEEEE;
}
.nombreProducto
{
       width: 300px;
       float: left;
       cursor: pointer;
}

.precioProducto
```

```
{
	width: 70px;
	font-weight: bold;
	text-align: right;
	float: left;
}

.agregarProducto
{
	width: 70px;
	font-weight: bold;
	cursor: pointer;
	color: Purple;
	float: right;
	text-align: right;
}

.agregarProducto:hover
{
	width: 20%;
	font-weight: bold;
	color: Red;
}
```

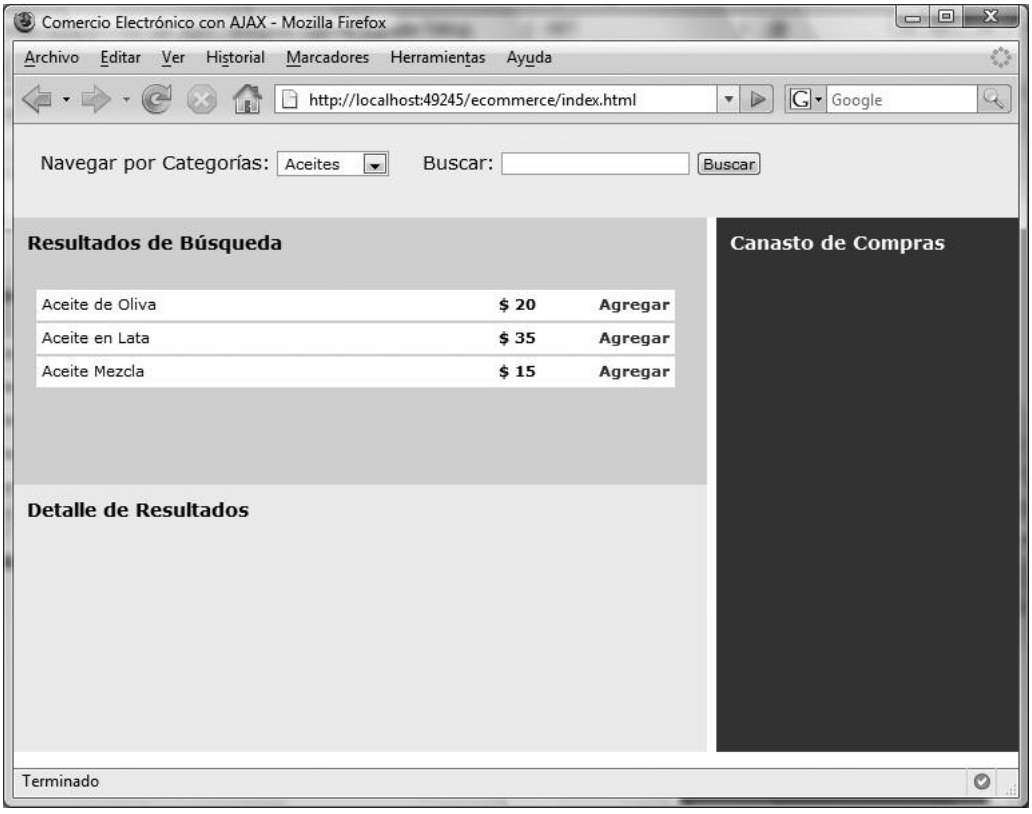

Fig. 7. Al elegir una categoría, se ve el listado de productos como resultado.

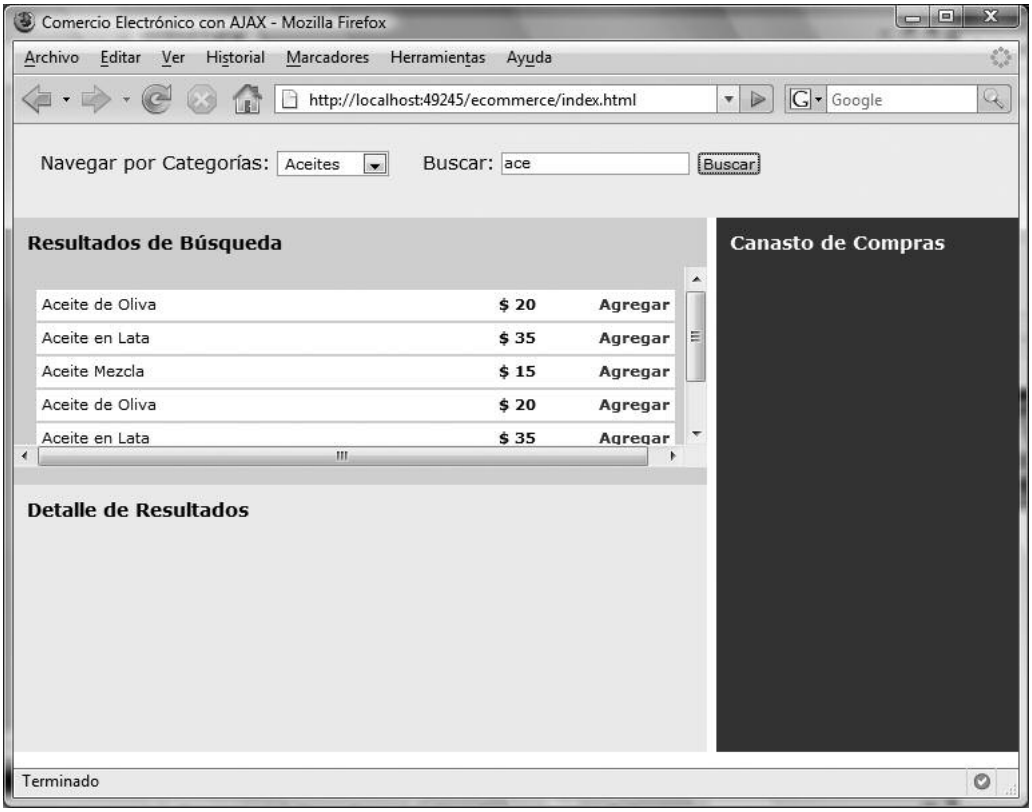

Fig. 8. CSS permite ubicarle una barra de desplazamiento a la lista de resultados cuando no entran en pantalla.

Se puede apreciar que cada resultado de la búsqueda está encapsulado en un DIV de clase productoResultado.

Cada fila tiene el nombre del producto, que al pulsar invoca la función detalle, el precio y un texto con apariencia de link que permite agregar el producto al canasto de compras, llamando a la función agregar. Ambas funciones todavía no están implementadas.

Nótese que mediante CSS se le dio cursor:pointer tanto al nombre como al link de agregar, para dar al usuario la idea de que puede utilizar esas opciones.

Detalle de producto

Cuando se selecciona un producto del resultado de la búsqueda, la idea es que debajo aparezca el detalle de ella, que emule un sistema de ventanas estilo Outlook donde se observa la vista previa de un producto.

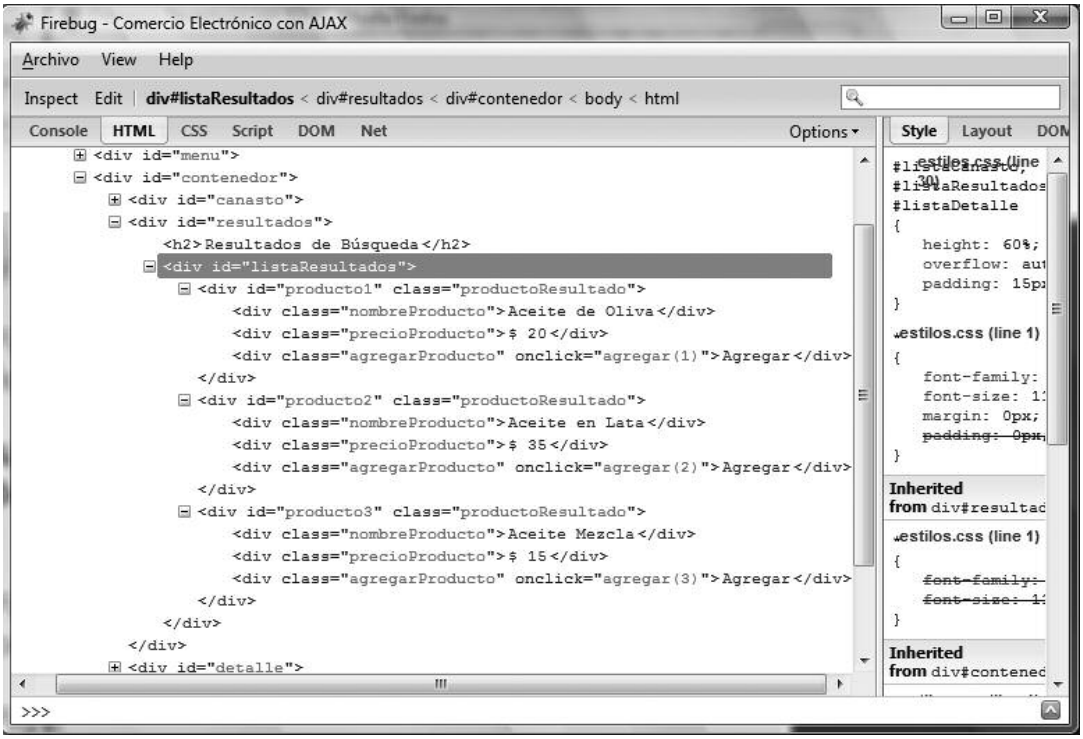

Fig. 9. Por medio del HTML Inspector de Firebug se puede ver cómo quedó la lista de resultados dinámica.

Para ello se tenía preparada la zona de Detalle. Lo que se debe hacer es, cuando se hace clic en un producto, ir a buscar sus datos completos y ubicarlos en esta zona.

Veamos el código que se agrega al index.js:

```
function detalle(id) {
    // Tengo que ir a buscar al servidor los detalles del producto
    $Ajax("producto.aspx?id=" + id, {
        onfinish: mostrarDetalle,
        tipoRespuesta: $tipo.JSON,
        avisoCargando: "cargando"
    });
}

function mostrarDetalle(producto) {
    // Creo el contenido de la zona de Detalle
    var html = "<div class='detalleNombre'>" + producto.nombre +
                    "</div>";
    if (producto.foto!= "") {
        // Si tiene foto, la incluimos
```

```
        html += "<img src='fotos/" + producto.foto + " '
                    class='detalleFoto' />";
    }
    html += "<input type='button' class='detalleAgregar' value='Agre
                    gar al Canasto' "
            + "onclick='agregar(" + producto.id + ")' />";
    html += "<div class='detalleCategoria'>Categoría: " +
                    producto.categoria + "</div>";
    html += "<div class='detallePrecio'>Precio: $" + producto.precio+
                    "</div>";
    html += "<div class='detalleDescripcion'>q" + producto.descrip
                    cion + "</div>";
    $("listaDetalle").innerHTML = html;
}
```

En este archivo se llama a un nuevo script de servidor que traerá un JSON con el detalle completo de un producto:

Producto.aspx

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">


    " ' <summary>
    " ' Devuelve el detalle de un Producto
    " ' </summary>
    " ' <remarks></remarks>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As Sys-
tem.EventArgs)
        'Este evento se ejecuta cuando se carga la página en el
        servidor
        Dim Resultado As DataTable

        'Leo el ID desde un parámetro GET
        'Convierto a Entero el ID para evitar problemas de seguridad
        Dim id As Integer
        Try
            id = Integer.Parse(Request.QueryString("id"))
        Catch ex As Exception
            id = 0
        End Try
        If id > 0 Then
            Dim sql As String
            sql = " SELECT PRODUCTOS.*, CATEGORIAS.NOMBRE AS CATEGO-
RIA "
```

```
            sql += " FROM PRODUCTOS INNER JOIN CATEGORIAS "
            sql += " ON IdCategoria = CATEGORIAS.Id WHERE "
            sql += " PRODUCTOS.Id = " + id.ToString
            Resultado = BD.Execute(sql)
            Dim Json As String = ""

            'Tomo la primera fila como resultado
            'dado que es el único registro
            Dim Registro As DataRow = Resultado.Rows(0)
            If Resultado.Rows.Count = 1 Then
                'Voy armando el JSON de cada producto
                Json += "{"
                Json += "id: " + Registro("id").ToString + ", "
                Json += "nombre: '" + Registro("nombre") + " ', "
                Json += "precio: " + Registro("precio").ToString + ",
"
                Json += "descripcion: '" +
                         Registro("descripcion").ToString + " ', "
                Json += "foto: '" + Registro("foto").ToString + " ', "
                Json += "categoria: '" + Registro("categoria").ToS-
                         tring + " ' "
                Json += "}"

            End If


            Response.Write(Json)

        End If
    End Sub
</script>
```

Por último se agregaron algunos estilos CSS para el detalle del producto (fig. 10):

```
.detalleNombre
{
      font-size: 19px;
      font-weight: bold;
      color: Blue;
}

.detalleAgregar
{
      float: right;
}

.detalleDescripcion
```

```
{
	font-style: italic;
	color: Gray;
}

.detallePrecio
{
	font-size: 15px;
	padding: 15px;
}

.detalleCategoria
{
	color: red
}

.detalleFoto
{
	float: left;
	padding: 5px;
	top: 0px;
	max-height: 110px;
}
```

Fig. 10. Se observa la zona de Detalle en funcionamiento utilizando AJAX y CSS.

Canasto de compras

Sólo resta la funcionalidad del canasto de compras. Aquí se deben cumplir las funciones siguientes:

- Incorporar productos que el usuario agrega desde las zonas de Resultados y de Detalle.
- Aumentar o reducir la cantidad de un producto ya agregado.
- Eliminar un producto del canasto.
- Vaciar el canasto.
- Mostrar el total por facturar.
- Permitirle al usuario finalizar la compra.

Todas estas funciones se van a resolver en la zona de Canasto. Además, hay que decidir dónde almacenar el canasto como tal. Se va a crear un modelo de datos local en el cliente que almacene el canasto de compras. La estructura será un vector que, en cada posición, tenga un objeto con las propiedades: id, nombre, precio y cantidad. Éste será un objeto global llamado **canasto**.

Ahora bien, si la función agregar sólo recibe el id del producto que se desea incorporar, ¿de dónde se obtiene su nombre y su precio? Parece poco lógico hacer una nueva petición al servidor para ir a buscar esa información cuando en realidad ya estaba disponible con anterioridad cuando se trajo el resultado de la búsqueda.

Para resolver este problema hay varias soluciones posibles; una de ellas es guardar en un vector tipo **hash**, también conocido como asociativo, todos los productos que se van leyendo desde resultados de búsqueda. Esto sería algo similar a una memoria caché de los productos que alguna vez pasaron por el resultado de búsqueda.

Los vectores tipo hash mantienen como índice cualquier valor alfanumérico. En este caso se puede utilizar el ID del producto como índice y guardar un objeto con las propiedades nombre y precio adentro.

```
// Hash de productos previamente navegados
var cacheProductos = [];

function agregarResultado(producto) {
    var div = "<div class='productoResultado' " +
              "id='producto" + producto.id + " '>";
    div += "<div class='nombreProducto' onclick='detalle(" +
                  producto.id + ")' >" + producto.nombre + "</div>";
    div += "<div class='precioProducto'>$ " + producto.precio +
                  "</div>";
    div += "<div class='agregarProducto' onclick='agregar(" +
```

```
                    producto.id + ")'>Agregar</div>";
    div += "</div>"
    $("listaResultados").innerHTML += div;


    // Agregamos el producto al hash
    cacheProductos[producto.id] = {
        nombre: producto.nombre,
        precio: producto.precio
    }

}
```

Agregando al canasto

Entonces, ahora se puede crear la función agregar:

```
// Agrega un producto al canasto de compras
function agregar(id) {
    // Primero buscamos si el producto ya estaba en el canasto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
        } else {
            i++;
        }
    }

    if (encontrado) {
        // El producto ya estaba en el canasto, sólo aumentamos
        // su cantidad. "i" tiene la posición actual
        canasto[i].cantidad++;
    } else {
        // El producto no estaba en el canasto, lo agregamos
        // obtenemos los datos del producto desde el hash global
        var producto = cacheProductos[id];
        nuevoProducto = {
                         'id': id,
                         'nombre': producto.nombre,
                         'precio': producto.precio,
                         'cantidad': 1
                        };

        canasto[canasto.length] = nuevoProducto;
    }
    // Llama a la función que dibuja el canasto
    actualizarCanasto();
}
```

La función realiza los siguientes pasos:

1. Busca primero si el producto agregado al canasto no se había agregado ya.
   a. En caso positivo, lo que hace es utilizar la posición encontrada en el canasto para aumentar la cantidad.
   b. En caso negativo, genera una entrada nueva en el canasto.
      i. Trae del hash global el producto buscando por ID.
      ii. Genera un objeto nuevo con los datos del producto y la cantidad en 1.
      iii. Agrega el objeto al array global del canasto.
2. Llama la función que actualiza en pantalla el canasto.

Entonces lo único que resta es generar la función que muestre el resultado en el canasto de compras. Se crea una función que recorre el vector del canasto y lo va dibujando sobre la pantalla, y otra que dibuja el total.

Mostrando el canasto

```
// Dibuja el canasto en la zona especificada
function actualizarCanasto()  {
    // Primero creamos la lista de productos
    var contenido = "";
    if (canasto.length==0) {
        contenido = "El canasto está vacío. <br />";
    } else {
        var filaCanasto;
        var total = 0;
        for (var i=0; i<canasto.length; i++) {
            // Creo una fila por cada producto
            filaCanasto = "<div class='filaCanasto' ";
            filaCanasto += " id='canasto_" + canasto[i].id + " '>";
            // Armamos cada fila con una tabla
            var tablaCanasto;
            tablaCanasto = "<table width='100%'><tr>";
            tablaCanasto += "<td class='canastoNombre'>" +
                                canasto[i].nombre + "</td>";
            tablaCanasto += "<td class='canastoCantidad'>Cant: " +
                                canasto[i].cantidad + "</td>";
            tablaCanasto += "<td class='canastoPrecio'>$" +
                                canasto[i].cantidad*canasto[i].precio +
                                "</td>";
            tablaCanasto += "<td><a href='javascript:quitar(" +
                                canasto[i].id
                + ")' class='canastoQuitar'>Quitar</a></td>";
            tablaCanasto += "</tr></table>";
            filaCanasto += tablaCanasto + "</div>";
```

```
            contenido += filaCanasto;
            total += canasto[i].precio * canasto[i].cantidad;
        }
        // Mostramos el TOTAL
        contenido += generarFilaTotal(total);

        // Agregamos el botón para Finalizar la Compra
        contenido += "<input type='button' id='btnFinalizar'
                        value='Finalizar Compra' />";
    }
    $("listaCanasto").innerHTML = contenido;
}

function generarFilaTotal(total) {
    var filaCanasto = "<div class='totalCanasto'> ";
    var tablaCanasto;
    tablaCanasto = "<table width='100%'><tr>";
    tablaCanasto += "<td class='canastoNombre'>TOTAL</td>";
    tablaCanasto += "<td class='canastoCantidad'></td>";
    tablaCanasto += "<td class='canastoPrecio'>$" + total+ "</td>";
    tablaCanasto += "<td><a href='javascript:vaciar()'
                        class='canastoQuitar'>Vaciar</a></td>";
    tablaCanasto += "</tr></table>";
    filaCanasto += tablaCanasto + "</div>";
    return filaCanasto;
}
```

Mediante CSS se le dan distintos estilos a cada elemento agregado:

```
.filaCanasto
{
    background: #000099;
    margin: 1px;
    padding: 2px;
}

.totalCanasto
{
    background: #990000;
    font-weight: bold;
    font-size: large;
}

.canastoNombre
{
    width: 110px;
}

.canastoPrecio
{
    width: 40px;
    text-align: right
```

```
}

.canastoCantidad
{
    width: 70px;
    text-align: right
}


.canastoQuitar
{
    color: White;
    font-size:smaller;
    text-align: right;
    padding-left: 2px;
}

#btnFinalizar
{
    margin-top: 15px;
    font-size: larger;
    background-color: White;
}
```

Ya se puede ver cómo queda el canasto de compras. Nótese que si se agrega un producto más de una vez, en lugar de sumarse una fila nueva se le cambia la cantidad al producto agregado con anterioridad (figs. 11 y 12).

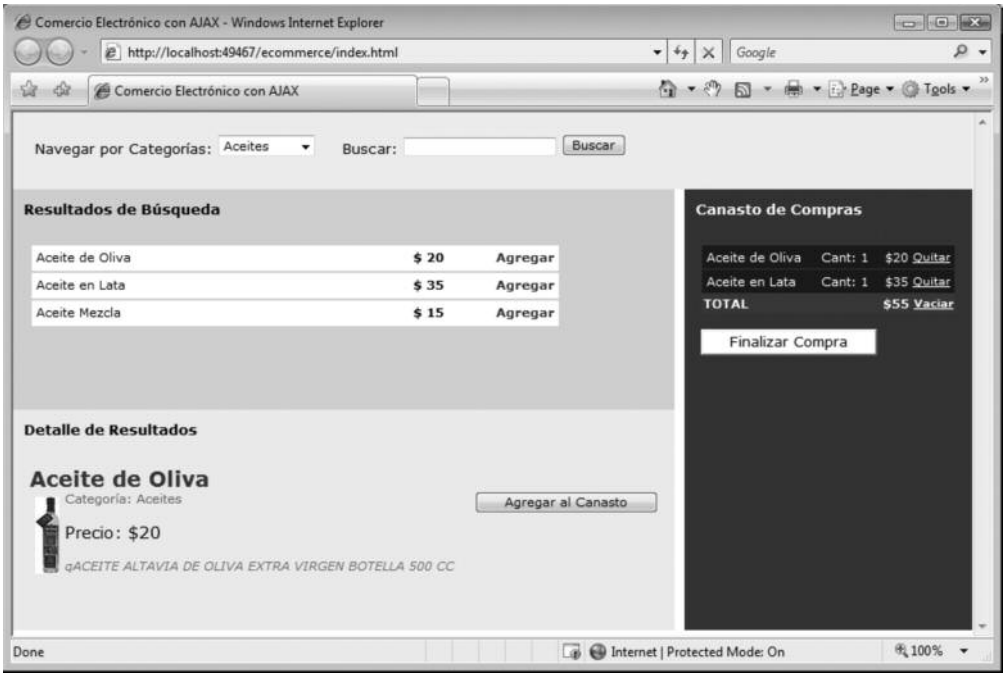

Fig. 11. El canasto de compras en funcionamiento.

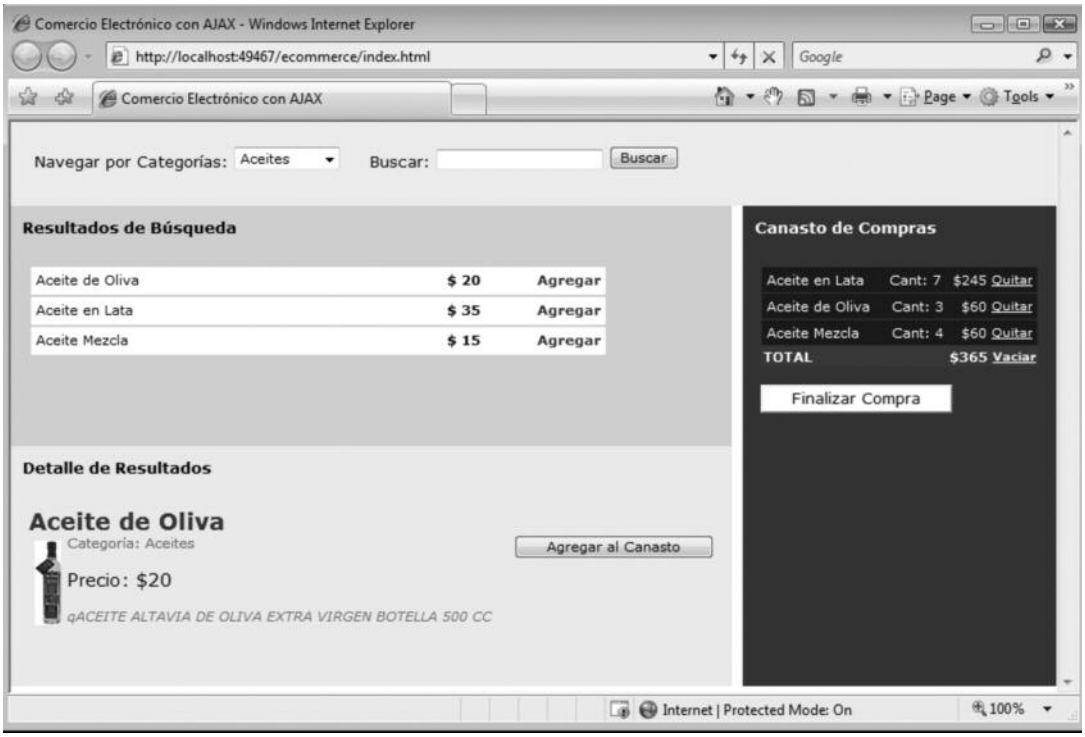

Fig. 12. Cuando se agrega un producto más de una vez se puede apreciar que se cambia su cantidad.

Quitando del canasto

Ahora queda implementar las funciones de quitar y vaciar, que eliminan un producto en particular o todos a la vez, respectivamente. La función de finalizar la compra se dejó para más adelante, en este capítulo.

Para quitar un elemento de un vector se utiliza la función without que incorpora la librería Prototype a todos los Arrays (fig. 13).

```
// Quita un producto del canasto
function quitar(id) {
    // Primero buscamos si el producto ya estaba en el canasto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
            // Lo eliminamos de la lista con Prototype
            canasto = canasto.without(canasto[i]);
        } else {
```

```
            i++;
        }
    }
    actualizarCanasto();
}

// Vacía todo el canasto
function vaciar() {
    canasto = [];
    actualizarCanasto();
}
```

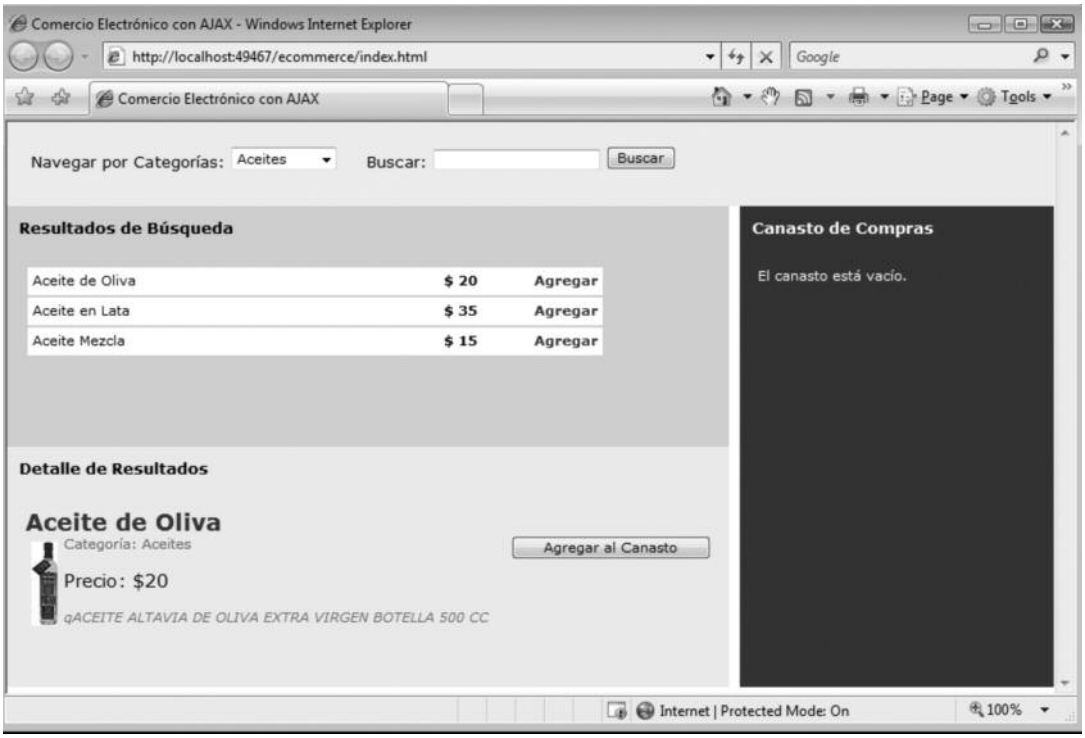

Fig. 13. Al quitar todos los productos o vaciar todo el canasto, se observa la leyenda que indica que no hay productos en el canasto.

Modificando cantidades

Hasta ahora sólo se puede modificar la cantidad de un producto dado buscándolo por categoría o nombre y agregando una cantidad de nuevo, y no es posible disminuir la cantidad. Sólo quitar todo el producto en cuestión. Falta la posibilidad de modificar la cantidad de un producto ya agregado en el canasto.

Para ello hay varias metodologías. Una sería implementar un editor en el mismo lugar o In place editor. Para lograrlo, se debe ocultar el div que muestra el texto y crear un campo de texto que, al salir de foco, se vuelve a convertir a un div.

Otra de las soluciones sería incluir dos imágenes (flecha arriba y flecha abajo) que permitan aumentar o disminuir los valores junto a cada cantidad. Se implementará esta opción utilizando dos archivos GIF para cada flecha: flechaup.gif y flechadown.gif.

Lo que se hace, entonces, es modificar el código para cuando se mostraba la cantidad con el código siguiente:

```
tablaCanasto += "<td class='canastoCantidad'>  " +
            canasto[i].cantidad
    + "<img src='flechadown.gif' onclick='bajar(" + canasto[i].id +
            ")' />"
    + "<img src='flechaup.gif' onclick='subir(" + canasto[i].id + ")'
            /></td>";
```

Asimismo, se agregan las dos funciones para subir y bajar la cantidad (figs. 14 y 15).

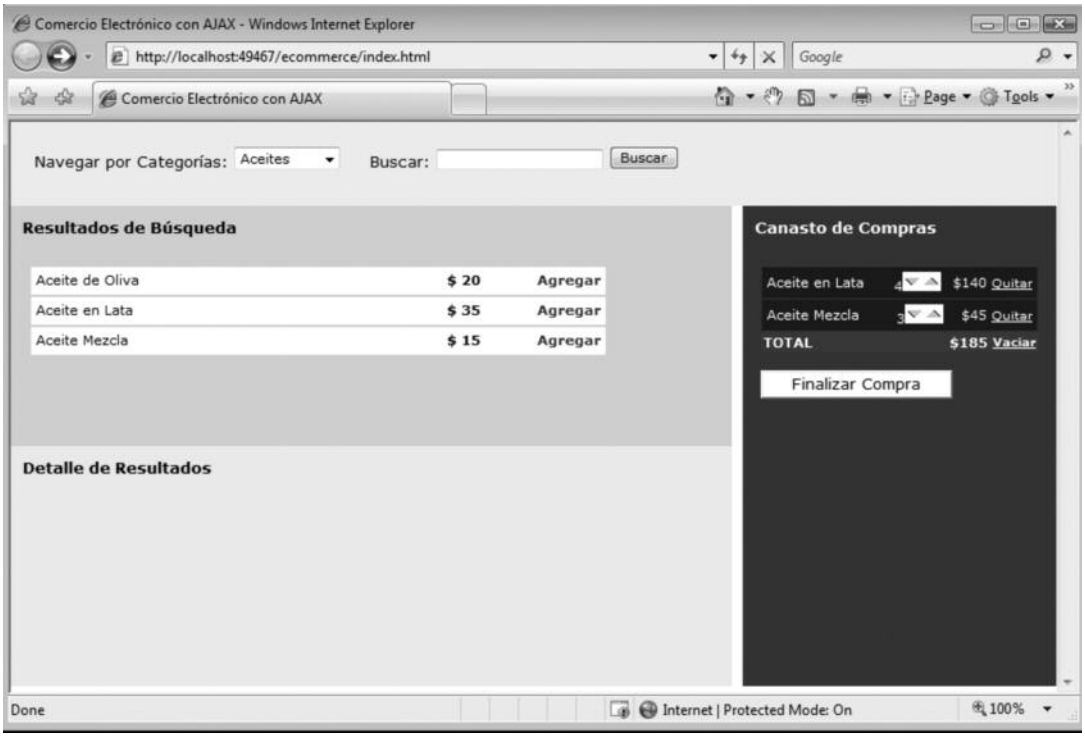

Fig. 14. Ahora se pueden modificar las cantidades mediante las flechas que tiene cada producto.

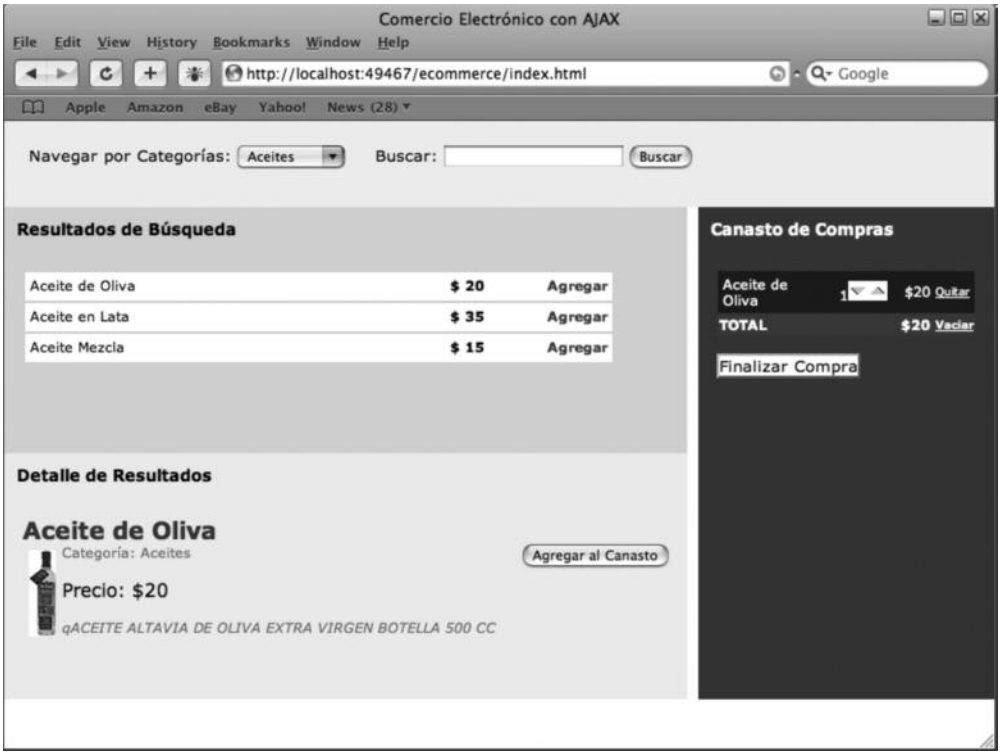

Fig. 15. Se observa el canasto de compras funcionando en un navegador Apple Safari, el clásico navegador de Mac, pero instalado sobre Windows.

Arrastrar y soltar

Introducción

Todos conocen el concepto de arrastrar y soltar (drag and drop, en inglés). Es la posibilidad de mover o copiar objetos en una interfaz gráfica arrastrándolos con el puntero y soltándolos en el destino.

Por defecto XHTML no provee metodología interna alguna para crear este efecto, aunque ofrece algunos eventos, como ondrop y ondrag desde JavaScript.

La librería Script.aculo.us, que ya conocemos, provee un entorno para trabajar con la modalidad de arrastrar y soltar de manera muy simple, sencilla, pero a la vez muy potente.

Para que esta funcionalidad esté disponible sólo se necesita contar con el archivo dragdrop.js, además del scriptaculous.js, en la carpeta de nuestro proyecto.

Scriptaculous trabaja con dos tipos de objetos: Draggables (zonas que se pueden arrastrar) y Droppables (zonas donde se pueden soltar elementos). Veamos cómo funciona el sistema. Una extensión a este framework es el objeto Sortables, ya utilizado con anterioridad para crear listas que se pueden reordenar.

Draggables

La idea es poder convertir todos los objetos que se desee que se puedan arrastrar en objetos Draggables. En principio cualquier elemento XHTML se puede convertir en un objeto que se puede arrastrar, aunque lo más común, y lo que trae menores problemas, es el conocido DIV (fig. 16).

La sintaxis es la siguiente:

```
new Draggable('id_del_elemento', opciones);
```

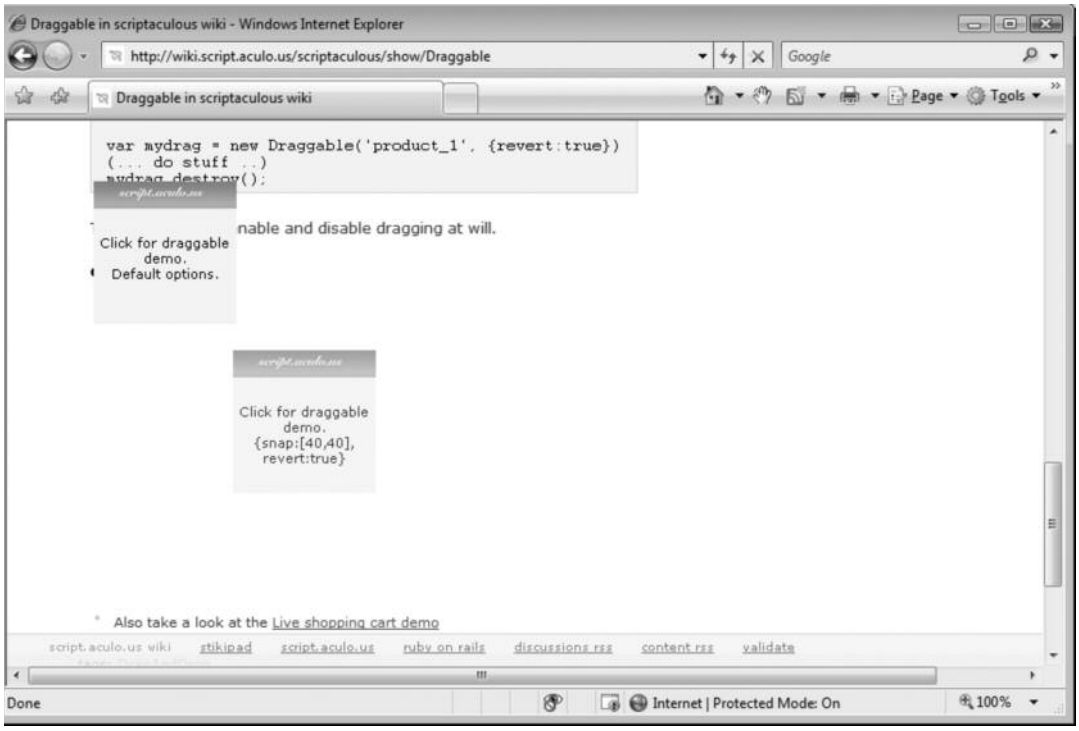

Fig. 16. En el sitio web de Script.aculo.us hay varios ejemplos de cómo implementar arrastrar y soltar.

Las opciones posibles son:

Propiedad	Descripción
revert	Si se define en true, cuando el usuario suelta el elemento, éste vuelve a su posición original con una animación
snap	Se define en false si se desea que el usuario pueda mover el objeto con libertad, o un vector [x, y] si lo que se quiere es que el usuario sólo pueda mover el elemento en una grilla de tamaño x,y
zindex	Es el valor CSS del eje Z que se aplicará al elemento cuando se está arrastrando
constraint	Se puede definir en horizontal o vertical para que sólo se pueda arrastrar en un eje, o vacío si se desea libertad
ghosting	Si se define en true, al comenzar a arrastrar el elemento se crea un clon del elemento y se lo arrastra, dejando el original en su lugar
starteffect	Define el efecto que se aplicará al comenzar a arrastrar. Por defecto es Opacity
reverteffect	Define el efecto que se aplicará cuando se suelta un elemento y éste vuelve a su posición original. Por defecto es Move
endeffect	Define el efecto que se aplicará cuando se suelta el elemento. Por defecto es Opacity
change	Es una función que se ejecutará ante cada cambio en el uso de arrastrar y soltar. Recibe el objeto Draggable donde se puede consultar en qué lugar está el objeto en la actualidad

Por defecto, al activar un objeto Draggable se lo podrá mover por toda la pantalla, pero no hay zonas específicas para soltar los elementos. Por defecto el elemento puede soltarse en cualquier lugar, por ejemplo:

```
new Draggable('divContenido');
```

Y si se desea que al soltarlo vuelva a su lugar original:

```
new Draggable('divContenido', {revert:true});
```

Por defecto el elemento, en este caso divContenido, siempre podrá arrastrarse. Para eliminar esta función, es preciso guardar el objeto y luego destruirlo, por ejemplo:

```
var drag = new Draggable('divContenido', {revert:true});
(…)
drag.destroy();
```

Droppables

Los elementos Droppables son los que tienen un ancho y un alto definidos, donde se podrán soltar objetos Draggables. Es factible crear y eliminar zonas de depósito de elementos con el código siguiente:

```
Droppables.add('id_elemento', opciones);
```

Las opciones disponibles son:

Propiedad	Descripción
accept	Por defecto, las zonas Droppables aceptan cualquier Draggable. Si se define esta propiedad con un string, o con un vector de strings, la zona sólo aceptará Draggables que tengan como className de CSS alguno de los strings ingresados
containment	Si se define esta propiedad con un ID o un vector de ID, la zona sólo aceptará elementos contenidos en esos ID (que sean descendientes)
hoverclass	Si se especifica, la zona Droppable tomará esta clase CSS cuando un elemento que se pueda aceptar se arrastre por encima de la zona
overlap	Se puede definir en horizontal o vertical. En ese caso sólo acepta el elemento cuando está más del 50% dentro de la zona en la dirección indicada

Si hay muchos objetos que se arrastran, se puede definir una clase CSS (aunque no tenga su definición de estilos) y agregársela a los elementos. Esto permitirá definirlos en la propiedad accept. Por otro lado, es una buena práctica definir un hoverclass que modifique levemente el color de fondo y aporte un borde resaltado a las zonas Droppables. Esto ayudará al usuario a darse cuenta dónde puede soltar los objetos.

También hay dos eventos disponibles que se pueden capturar:

Propiedad	Descripción
onHover	Se ejecuta cada vez que el usuario arrastra un elemento que se puede aceptar en la zona Droppable. Esta función recibe el objeto Draggable, el Droppable y el porcentaje de solapamiento actual
onDrop	Se ejecuta cuando el usuario suelta un Draggable sobre la zona y es un elemento que se puede aceptar. Recibe el objeto Draggable, el Droppable y un objeto Event con algunas propiedades útiles

Por medio del evento onDrop se puede capturar el momento en que el usuario "suelta" un elemento y actuar en consecuencia.

La gran pregunta que surge al comienzo del uso de Drag and Drop en Scriptaculous es cómo se identifica el objeto que se soltó en la zona Droppable. Para eso se utiliza el primer parámetro del evento onDrop, que da el elemento Draggable y se puede obtener cualquier propiedad de él, como ID o alt.

El tercer parámetro que se recibe en la función onDrop posee las propiedades siguientes:

Propiedad	Descripción
type	Evento desencadenado (por ejemplo: mouseup)
target	Destino del drop
screenX, screenY	Posición X e Y de la pantalla donde se soltó el elemento
ctrlKey	Indica true si la tecla Control estaba presionada cuando soltó el elemento
shiftKey	Indica true si la tecla Shift estaba presionada cuando soltó el elemento
altKey	Indica true si la tecla Alt estaba presionada cuando soltó el elemento

Se puede eliminar una zona Droppable con:

```
Droppables.remove('id_elemento');
```

Uso avanzado

Es posible trabajar con arrastrar y soltar en un nivel más avanzado por medio del objeto Draggables disponible en JavaScript por la librería.

Este objeto tiene los siguientes métodos y propiedades:

Propiedad	Descripción
drags	Es un vector con todos los elementos Draggables actuales del documento
observers	Es un array con todos los observadores
register(función)	Registra una función que recibe un Draggable. Se ejecutará cada vez que se agregue un elemento que se puede arrastrar
unregister(función)	Elimina la función de la lista
activate(draggable)	Activa la opción de arrastrar sobre un elemento aunque el usuario no lo haya iniciado
deactivate(draggable)	Desactiva el arrastre de un elemento
addObserver(obs)	Agrega un objeto observador
removeObserver(obs)	Elimina un objeto observador
notify()	Notifica a todos los observadores

Un observador es un objeto que tiene definido element, con el elemento que se desea observar, y una o más de las siguientes funciones definidas: onStart, onDrag, onEnd. Esto permite centralizar el control de los elementos que se arrastran por la página web.

Aplicación

Ahora se van a aplicar todos los conceptos vistos antes al proyecto de comercio electrónico.

Arrastrando al canasto

Lo primero en que se piensa es en implementar la función de arrastrar y soltar entre los elementos del resultado de la búsqueda y el canasto de compras.

Para ello, hay que elegir qué elementos van a ser Draggables y cuáles van a ser Droppables. Para eso se puede analizar cómo estaba organizada la estructura de elementos en nuestro proyecto, como se puede ver en la figura 17.

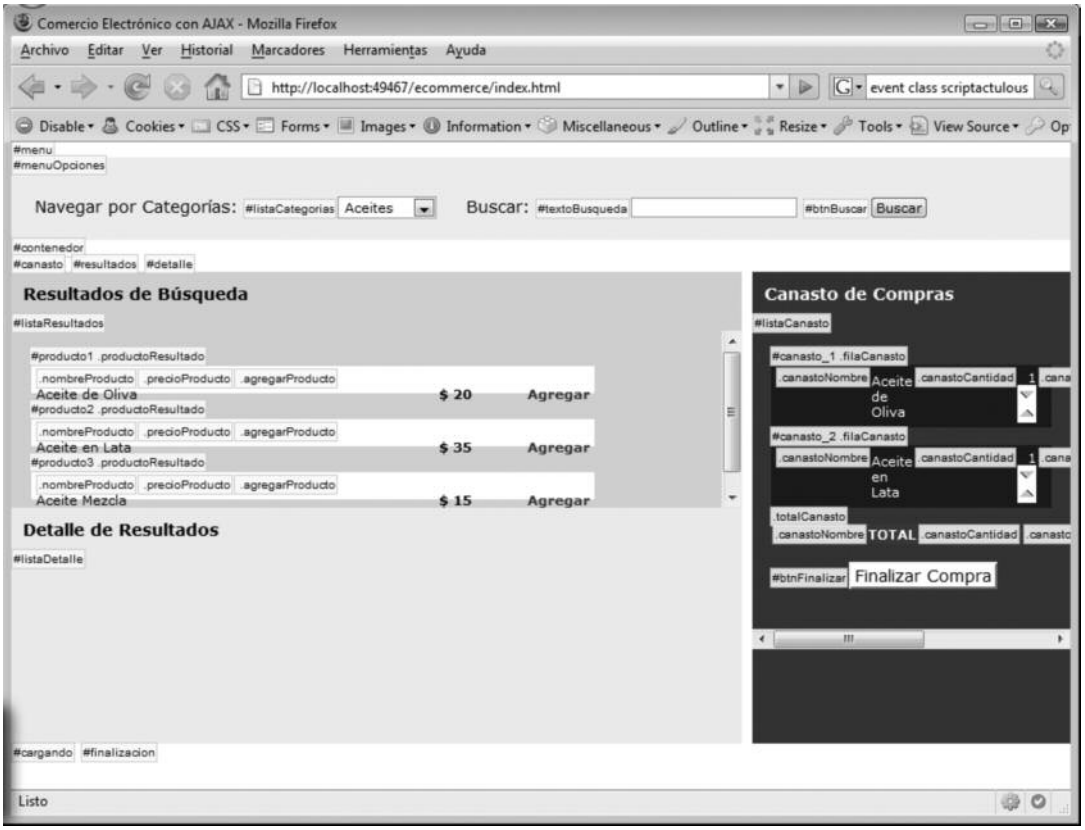

Fig. 17. Es factible analizar qué elementos son Draggables y Droppables utilizando las herramientas que se pueden anexar a los navegadores.

Al analizar allí, se puede decidir que el elemento con ID listaCanasto debería ser nuestro elemento Droppable, y los elementos que se pueden arrastrar y soltar allí deberían ser los de clase CSS productoResultado (p. ej., los ID producto1 y producto3).

El elemento Droppable se puede definir directamente en el window.onload debido a que, aunque todavía no haya elementos, debería ser posible arrastrar los resultados de búsqueda. Ya es factible definir que sólo se aceptan elementos que sean de clase productoResultado y así restringir el acceso a la zona del canasto.

```
window.onload = function() {
    // …
```

```
    // Define el Droppable
    Droppables.add("listaCanasto", {
        onDrop: sueltaProducto,
        accept: "productoResultado"
    });
}
```

Queda por definir la función sueltaProducto. ¿Cómo se sabe qué producto se acaba de soltar en el canasto? Para eso se pueden utilizar los ID de los elementos Draggables que son del estilo: productoXX, donde XX es el ID del producto en la tabla. Entonces, mediante los métodos de la clase String se puede obtener el ID que hay que agregar.

```
// Se ejecuta cuando el usuario suelta un producto en el canasto
function sueltaProducto(drag, drop, evento) {
    // Obtenemos el ID
    var id = drag.id.subString(9, drag.id.length);
    // Agregamos el producto al canasto
    agregar(id);
}
```

Los elementos Draggables se deben definir cuando se crean, y esto se hacía cuando se recibían los resultados de la búsqueda del servidor.

Si se presentan algunos problemas con el contenedor cuando se arrastra un elemento (el elemento se queda en su contenedor), se debe eliminar la propiedad overflow del div contenedor.

Veamos el código:

```
function agregarResultado(producto) {
    var div = document.createElement("div");
    div.className = 'productoResultado';
    div.id = 'producto' + producto.id;
    div.innerHTML += " <div class='nombreProducto' onclick='detalle("
            + producto.id + ")' >" + producto.nombre + "</div>";
    div.innerHTML += " <div class='precioProducto'>$ " +
            producto.precio + "</div>";
    div.innerHTML += " <div class='agregarProducto' onclick='agre-
            gar(" + producto.id + ")'>Agregar</div>";
    $("listaResultados").appendChild(div);
    // Agregamos el producto al hash
    cacheProductos[producto.id] = {
        nombre: producto.nombre,
```

```
            precio: producto.precio,
            // Definimos el producto como Draggable
            drag: new Draggable("producto" + producto.id, {revert:true})
        }
    }
```

Se agrega al cacheProductos para luego tener una referencia al objeto Draggable (fig. 18).

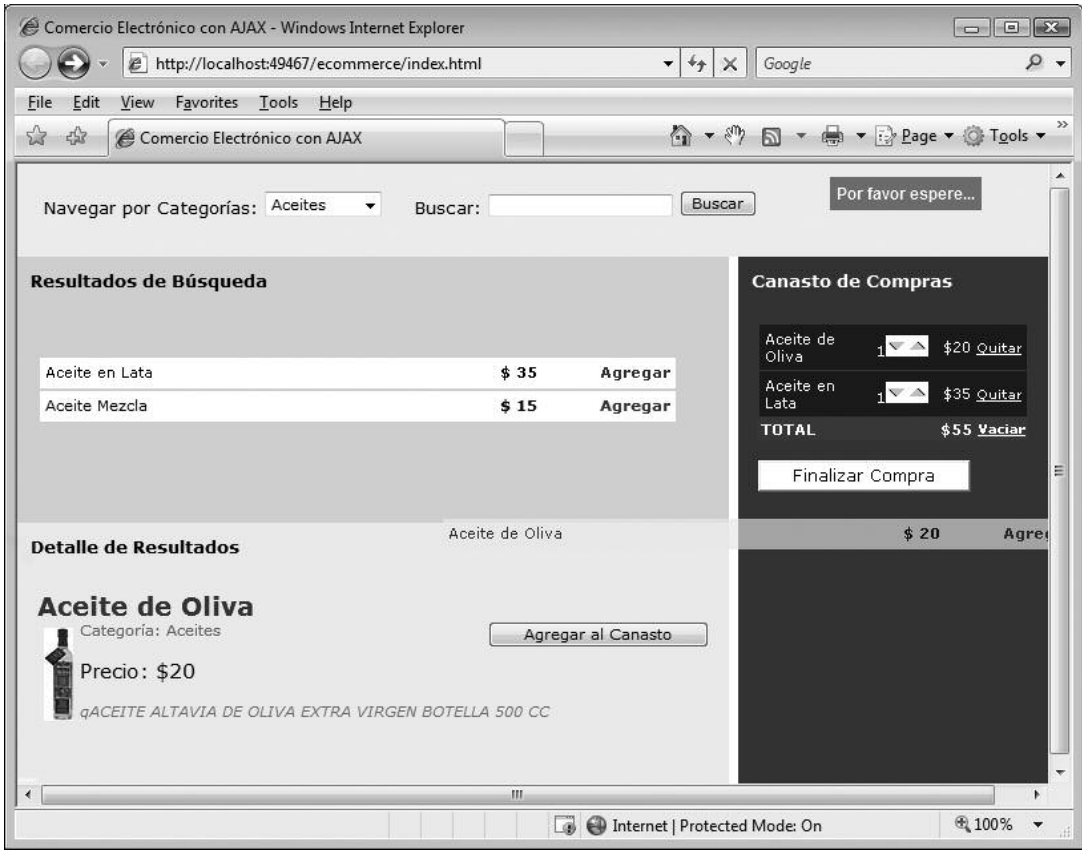

Fig. 18. Cuando se utiliza revert:true, al soltar el elemento éste vuelve a su posición original, pero con el evento onDrop se lo agrega al canasto.

También es factible hacer que se pueda arrastrar la foto del producto desde el detalle. Para ello se cambia la función siguiente:

```
function mostrarDetalle(producto) {
    // Creo el contenido de la zona de Detalle
    var html = "<div class='detalleNombre'>" + producto.nombre +
                  "</div>";
    if (producto.foto!= "") {
        // Si tiene foto, la incluimos
        html += "<img src='fotos/" + producto.foto + " ' class='deta-
                  lleFoto' "
             + "id='foto" + producto.id + " ' />";
    }
    html += "<input type='button' class='detalleAgregar' value='Agre-
                  gar al Canasto' "
            + "onclick='agregar(" + producto.id + ")' />";
    html += "<div class='detalleCategoria'>Categoría: " +
                  producto.categoria + "</div>";
    html += "<div class='detallePrecio'>Precio: $" + producto.precio+
                  "</div>";
    html += "<div class='detalleDescripcion'>q" + producto.descrip-
                  cion + "</div>";
    $("listaDetalle").innerHTML = html;

    new Draggable("foto" + producto.id, {revert:true});
}
```

Además, ahora también hay que aceptar en la zona de Drop la clase CSS detalleFoto, por lo que se cambia por:

```
window.onload = function() {
// …

    // Define el Droppable
    Droppables.add("listaCanasto", {
        onDrop: sueltaProducto,
        accept: ["productoResultado", "detalleFoto"]
    });
}
```

Asimismo, se debe cambiar la función sueltaProducto para tomar tanto fotos como filas del resultado de búsqueda (fig. 19):

```
// Se ejecuta cuando el usuario suelta un producto en el canasto
function sueltaProducto(drag, drop, evento) {
    var id;
```

```
    if (drag.id.substring(0, 8)==”producto”) {
        // Obtenemos el ID, es productoXXX
        id = drag.id.substring(8, drag.id.length);
    } else {
        // Obtenemos el ID, es fotoXXX
        id = drag.id.substring(4, drag.id.length);
    }
    // Agregamos el producto al canasto
    agregar(id);
    // Hacemos un efecto sobre el canasto
    // Para que el usuario vea la confirmación de su Drop
    new Effect.Appear('listaCanasto', {duration: 0.2});
}
```

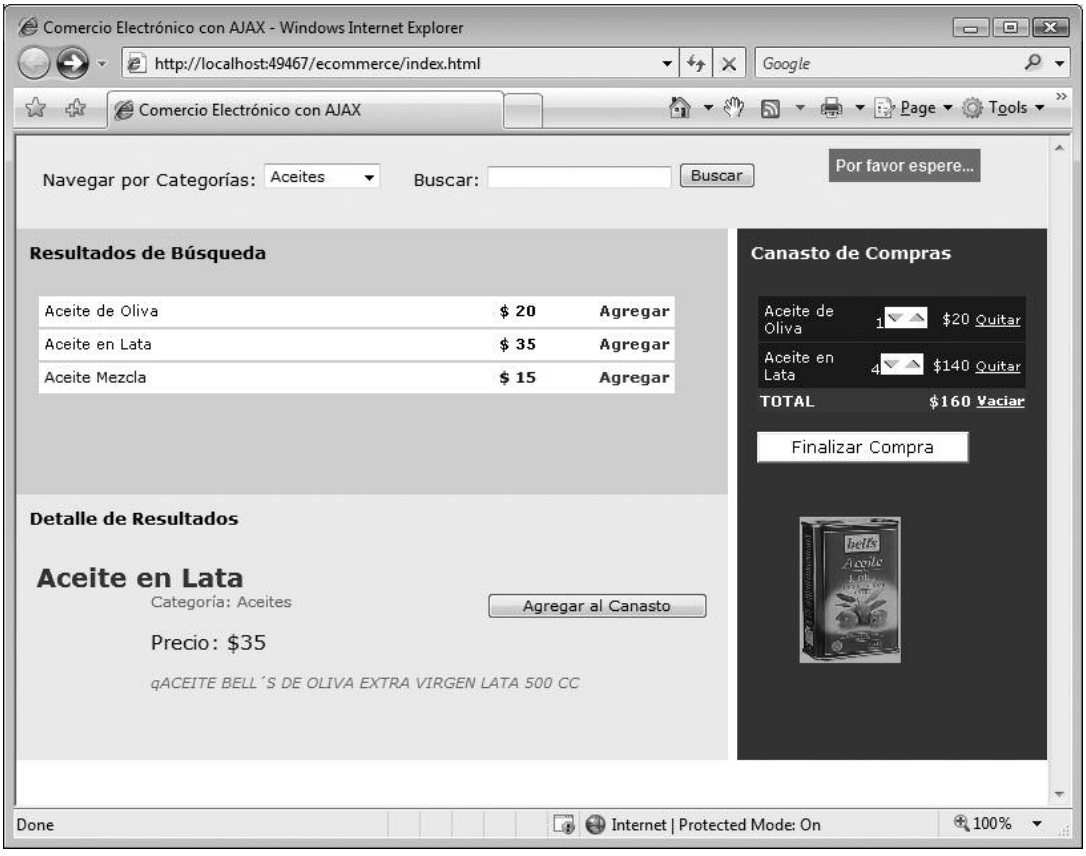

Fig. 19. Ahora también se pueden arrastrar las fotos del detalle hacia el canasto y se agrega el producto.

Arrastrando a la basura

Otra de las grandes ideas que se pueden implementar con arrastrar y soltar es incorporar un "bote de basura" en la zona del canasto y que se puedan arrastrar los productos que hay allí hacia el bote, lo que sería equivalente a utilizar la opción "Quitar".

En este caso, la zona de Drop es la misma imagen que se incorporó y los elementos Draggables son todas las filas que se incluyeron en el canasto que son de clase filaCanasto.

```
function actualizarCanasto() {
       // …

    // Agrego todas las filas del canasto como Draggables
    for (var i=0; i<canasto.length; i++) {
        new Draggable("canasto_" + canasto[i].id, {revert:true});
    }

    // Creamos el bote de basura
    var bote = document.createElement("img");
    bote.src = "basura.png";
    bote.width = "100";
    bote.height = "100";
    bote.id = "basura"
    $("listaCanasto").appendChild(bote);
    // Creo la zona de Drop con el bote
    Droppables.add("basura", {
        accept: 'filaCanasto',
        onDrop: tiraProducto
    });
```

Y la función tiraProducto hace lo siguiente (fig. 20):

```
}


function tiraProducto(drag, drop, evento) {
    var id = drag.id.substring(8, drag.id.length);
    quitar(id);
}
```

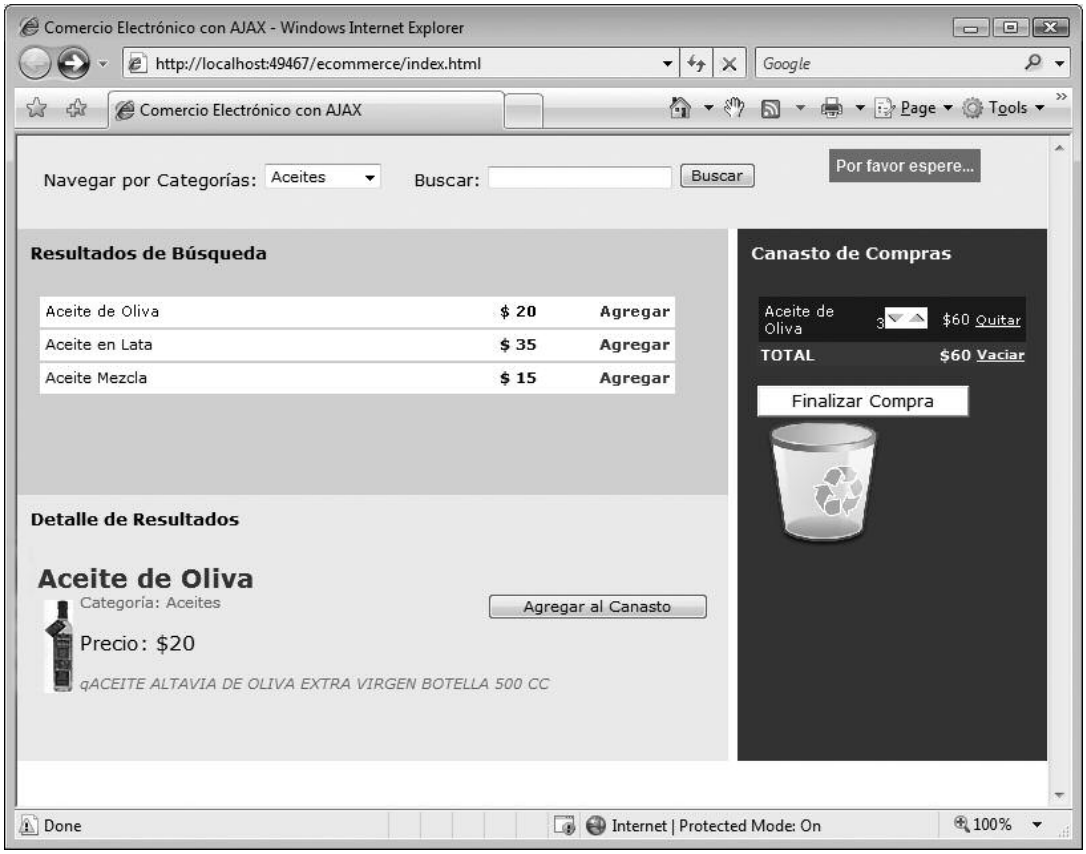

Fig. 20. Ahora hay dos tipos de objetos Drop, el canasto y el bote de basura, donde se pueden soltar los elementos del canasto.

Finalizando la compra

Ahora le toca el turno a la finalización de la compra. Se había colocado el botón de "Finalizar", pero que por el momento no hacía nada. Lo primero que se debe hacer es agregarle comportamiento:

```
$("btnFinalizar").onclick = finalizar;
```

El paso siguiente es armar el formulario con XHTML y CSS en el div que se había creado originalmente vacío (fig. 21):

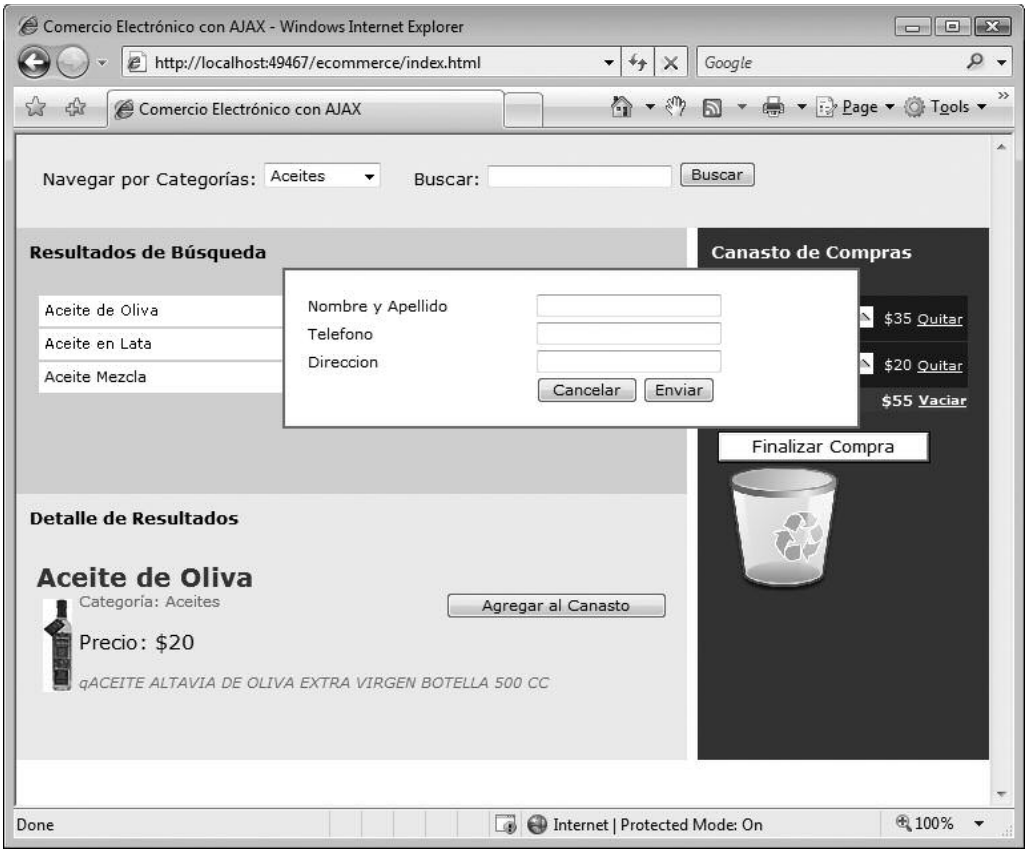

Fig. 21. El formulario en funcionamiento. Se podría aplicar un estilo para lograr que sea "modal" y, de esa forma, que no se pueda utilizar el resto de la página mientras esta ventana se encuentra vigente.

```
<div id="finalizacion">
    <table width="100%">
        <tr>
            <td>Nombre y Apellido</td>
            <td><input type="text" id="txtNombre" /></td>
        </tr>
        <tr>
            <td>Telefono</td>
            <td><input type="text" id="txtTelefono" /></td>
        </tr>
        <tr>
            <td>Direccion</td>
            <td><input type="text" id="txtDireccion" /></td>
        </tr>
        <tr>
            <td></td>
```

```
            <td>
                <input type=”button” id=”btnCancelar” value=”Cance-
lar” onclick=”cancelar()” />
                    <input type=”button” id=”btnEnviar” value=”En-
viar” onclick=”enviar()” />
            </td>
        </tr>
    </table>

</div>
```

```
#finalizacion
{
    display: none;
    width: 400px;
    top: 100px;
    left: 200px;
    position: absolute;
    background-color: White;
    border: 2px gray solid;
    padding: 15px;
}
```

Para crear un efecto de ventana modal, sólo se debe insertar el DIV dentro de otro con algún identificador, por ejemplo, ventanaModal:

```
<div id=”ventanaModal”>
      <div id=”finalizacion”>
            …
      </div>
</div>
```

Agregar el siguiente código al CSS

```
#ventanaModal
{
     visibility: hidden;
     position: absolute;
     left: 0px;
     top: 0px;
     width:100%;
```

```
     height:100%;
     text-align:center;
     z-index: 1000;
     background-image:url(fondo_modal.gif);
}

#ventanaModal div
{
     width:300px;
     margin: 100px auto;
     background-color: #fff;
     border:1px solid #000;
     padding:15px;
     text-align:center;
}
```

Para que funcione bien en Internet Explorer, es necesario definirle a body la siguiente propiedad CSS:

```
height:100%;
```

Mediante un GIF o un PNG transparente (fondomodal.gif) aplicado como fondo de la ventanaModal es posible dar un efecto de grisado sobre todo el fondo, para que el usuario comprenda que primero debe cerrar la ventana modal. Este fondo tendrá un tramado transparente. Ahora cuando se desee colocar la ventana sólo se deberá hacer visible ventanaModal (figs. 22 y 23).

La función finalizar queda:

```
// Muestra el formulario de Finalización de Compra
function finalizar() {
    $(”ventanaModal”).style.visibility=”visible”;
}
```

Luego queda la función de cancelar, que sólo quita el formulario:

```
// Cancela el envío de la compra
function cancelar() {
    $(”ventanaModal”).style.visibility=”hidden”;
}
```

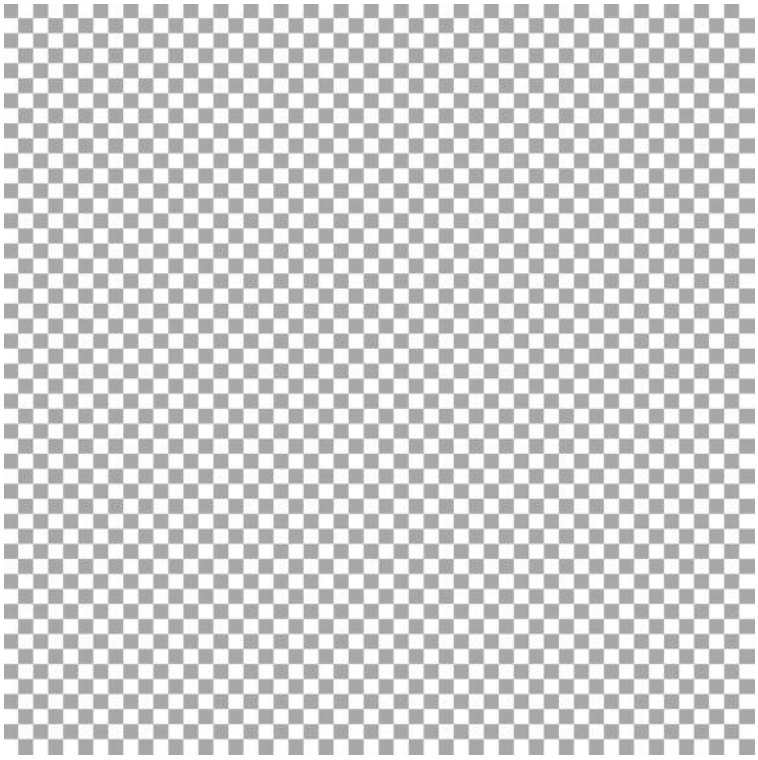

Fig. 22. Con un tramado transparente se puede lograr el efecto de la ventana modal.

Por último se encuentra la función que recolecta toda la información y la envía al servidor. Dejamos en el lector las validaciones pertinentes.

```
// Envía toda la información del pedido
function enviar() {
    // TODO: Validaciones
    // Armamos un string de tipo QueryString con los datos personales
    var parametros = "";
    parametros += "nomyape=" + $F("txtNombre");
    parametros += "&direccion=" + $F("txtDireccion");
    parametros += "&telefono=" + $F("txtTelefono");


    // Recorremos todo el canasto y armamos un string con los ID y
       las cantidades
    var strCarrito = "";
```

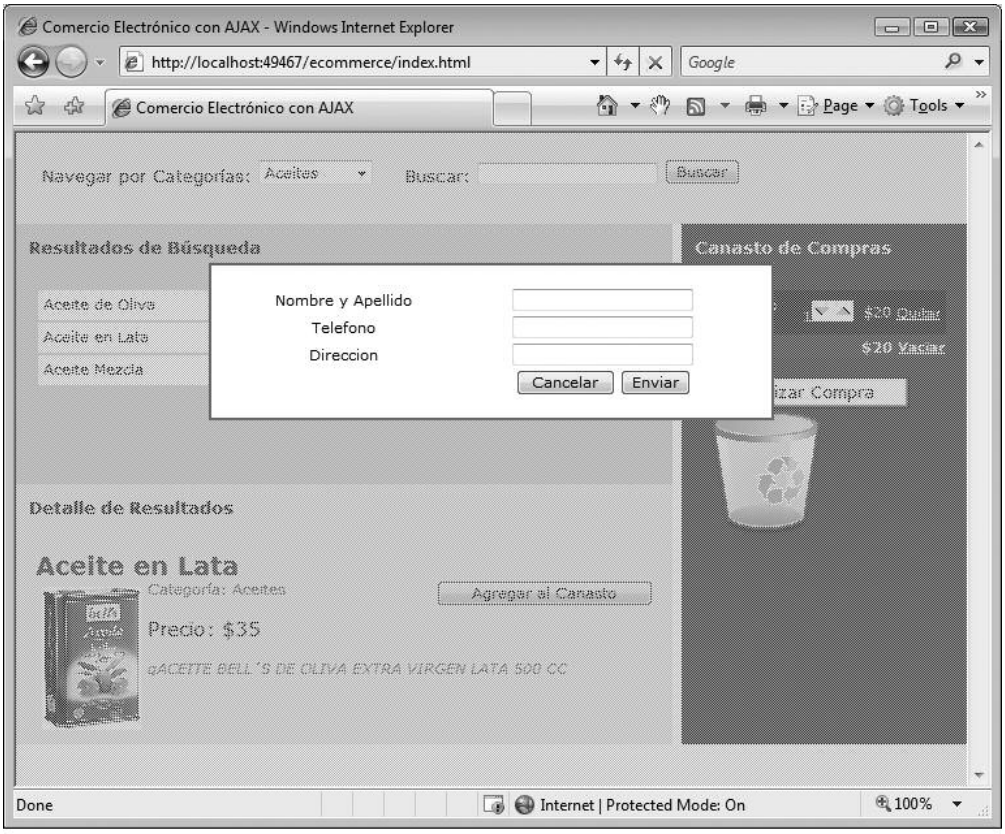

Fig. 23. Se observa la ventana modal. Nótese que todo el fondo está apagado y no puede utilizarse hasta tanto se cierre la ventana.

```
for (var i=0; i<canasto.length; i++) {
    strCarrito += canasto[i].id;
    // Para separar ID de Cantidad usamos un guion
    strCarrito += "-";
    strCarrito += canasto[i].cantidad;
    // Usamos punto para separar productos
    strCarrito += ".";
}
// Eliminamos último pipe
strCarrito = strCarrito.substring(0, strCarrito.length-1);
parametros += "&carrito=" + strCarrito;

$Ajax("enviar.aspx", {
    metodo: $metodo.POST,
    parametros: parametros,
```

```
        avisoCargando: "cargando",
        onfinish: function() {
            alert("Su pedido ha sido guardado");
            cancelar();
            vaciar();
        }
    });
}
```

Enviar.ASPX guarda el pedido en la base de datos. Éste tiene el código VisualBasic.NET siguiente:

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As Sys-
tem.EventArgs)
        'Faltan las Validaciones
        Dim nombre As String = Request.Form("nomyape")
        Dim direccion As String = Request.Form("direccion")
        Dim telefono As String = Request.Form("telefono")

        Dim sql As New SqlCommand
        'SELECT @@IDENTITY nos devuelve el último ID autonumérico
                insertado
        sql.CommandText = "INSERT INTO PEDIDOS (Fecha, Finalizado,
NombreApellido, Telefono, Direccion) VALUES (@Fecha, @Finalizado,
@NombreApellido, @Telefono, @Direccion); SELECT @@IDENTITY"
        With sql.Parameters
            .AddWithValue("@Fecha", Date.Today())
            If Request.Form("temporal") Is Nothing Then
                .AddWithValue("@Finalizado", True)
            Else
                .AddWithValue("@Finalizado", True)
            End If
            .AddWithValue("@NombreApellido", nombre)
            .AddWithValue("@Direccion", direccion)
            .AddWithValue("@Telefono", telefono)
        End With
        'Ejecuto un escalar para retener el ID insertado
        Dim id As Integer = BD.ExecuteScalar(sql)

        'Proceso el canasto de compras
        Dim canasto As String = Request.Form("carrito")
```

```
        'Genero un vector de Productos desde el string
        Dim vecProductos As String()
        vecProductos = canasto.Split(".")
        For Each producto As String In vecProductos
            'Separo cada producto de su cantidad
            Dim vecAux As String()
            vecAux = producto.Split("-")
            Dim idProducto As Integer = Integer.Parse(vecAux(0))
            Dim cantidad As Integer = Integer.Parse(vecAux(1))

            'Inserto el registro en la tabla detalle
            sql = New SqlCommand
            sql.CommandText = "INSERT INTO PRODUCTOSxPEDIDO (IdPro-
ducto, IdPedido, Cantidad) VALUES (@IdProducto, @IdPedido, @Canti-
dad)"
            With sql.Parameters
                .AddWithValue("@IdProducto", idProducto)
                .AddWithValue("@IdPedido", id)
                .AddWithValue("@Cantidad", cantidad)
            End With
            BD.ExecuteNonQuery(sql)

        Next

    End Sub
</script>
```

Guardado automático

Un servicio interesante que se le puede brindar al usuario es guardarle automáticamente el canasto de compras para futuros ingresos o ante el caso de que se le cierre el navegador o se le cuelgue la aplicación AJAX. De esta forma, la próxima vez que inicie el sistema se le podría avisar que tiene un canasto de compras guardado y ofrecer si quiere recuperarlo.

Este proceso debería hacerse automáticamente desde el momento en que el usuario agregue algo al canasto y, al cargar la página por primera vez, consultarle al servidor con otra petición si tiene un canasto guardado.

El canasto se puede guardar en:

- Una cookie, es un valor que queda almacenado en la computadora del usuario. También es posible guardar el ID del pedido en una cookie. De esta manera el servidor buscaría un pedido no finalizado con ese ID.
- En sesión en el servidor. Se mantendría en memoria del servidor mientras dure la sesión (por lo general hasta que pasen 20 minutos sin actividad).

Para ello simplemente se genera una versión del finalizar que envíe también el parámetro POST temporal, y el ASPX ya tiene configurado que guarde que no está finalizado cuando así sea. A esta función se la llama con un temporizador, por ejemplo, cuando el canasto tiene algo (en el agregar):

```
// Variables globales
var autoguardado;

…

function agregar() {
      …
      if (autoguardado==undefined) {
            autoguardado = setInterval(30000, ”autoguardado()”);
      }
}

function quitar() {
      …
      if (canasto.length==0) {
            // Me quedé sin productos
            clearInterval(autoguardado);
      }
}

function vaciar() {
      clearInterval(autoguardado);
}
```

Código completo

Ya se analizaron todas las cuestiones relativas al sitio de comercio electrónico. Dado que se fue modificando de a poco el proyecto con fines didácticos, véase cómo quedaron el XHTML, el CSS y el JS en versión final.

index.html

```
<!DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Transitional//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head>
      <meta http-equiv=”Content-Type” content=”text/html;
charset=iso-8859-1” />
```

```
        <title>Comercio Electrónico con AJAX</title>
        <link href="estilos.css" type="text/css" rel="stylesheet" />
        <script type="text/javascript" src="prototype.js"></script>
        <script type="text/javascript" src="scriptaculous.js"></script>
        <script type="text/javascript" src="AjaxLib.js"></script>
        <script type="text/javascript" src="index.js"></script>
</head>

<body>

        <div id="menu">
              <div id="menuOpciones">
                    Navegar por Categorías:
                    <select id="listaCategorias">
                    </select>

                    Buscar:
                    <input type="text" id="textoBusqueda" />
                    <input type="button" id="btnBuscar" value="Buscar" />
              </div>
        </div>

        <div id="contenedor">
              <div id="canasto">
                    <h2>Canasto de Compras</h2>
                    <div id="listaCanasto">
                    </div>
              </div>
              <div id="resultados">
                    <h2>Resultados de Búsqueda</h2>
                    <div id="listaResultados">
                    </div>
              </div>
              <div id="detalle">
                    <h2>Detalle de Resultados</h2>
                    <div id="listaDetalle">
                    </div>
              </div>
        </div>

        <div id="cargando">Por favor espere...</div>
        <div id="ventanaModal">
            <div id="finalizacion">
                <table width="100%">
                    <tr>
                        <td>Nombre y Apellido</td>
                        <td><input type="text" id="txtNombre" /></td>
                    </tr>
                    <tr>
                        <td>Telefono</td>
                        <td><input type="text" id="txtTelefono" /></td>
```

```
                    </tr>
                    <tr>
                        <td>Direccion</td>
                        <td><input type="text" id="txtDireccion" /></td>
                    </tr>
                    <tr>
                        <td></td>
                        <td>
                            <input type="button" id="btnCancelar"
value="Cancelar" onclick="cancelar()" />
                            <input type="button" id="btnEnviar"
value="Enviar" onclick="enviar()" />
                        </td>
                    </tr>
                </table>

            </div>
        </div>

</body>
</html>
```

estilos.css

```
* {
        font-family:Verdana, Arial, Helvetica, sans-serif;
        font-size: 11px;
        padding: 0px;
        margin: 0px;
}

#menu {
        width: 100%;
        background-color:#CCFF99;
        height: 70px;
}

#menuOpciones {
        height: 70px;
        vertical-align: middle;
        padding: 20px;
        font-size: larger;
}

#listaCategorias {
        margin-right: 20px;
```

```
}

#contenedor h2 {
	padding: 10px;
	font-size: larger;
}

#listaCanasto, #listaResultados, #listaDetalle {
	padding: 15px;
	/*overflow: auto;*/
	height: 60%;
}

#resultados {
	width: 69%;
	background-color: #FFCCCC;
	height: 200px;
	float: left;
}

#detalle {
	width: 69%;
	background-color: #EEEEEE;
	height: 200px;
	float: left;
}

#canasto {
	float: right;
	width: 30%;
	background-color:#0033CC;
	color: white;
	height: 400px;
}

#cargando {
	position: absolute;
	top: 10px;
	right: 50px;
	background-color: red;
	color: white;
	font-family: Arial, Helvetica, sans-serif;
	font-size: 12px;
	font-weight: bold;
	padding: 5px;
}

.productoResultado {
	width: 470px;
	padding: 4px;
	margin: 2px;
```

```
        background-color: White;
        font-size: large;
        height: 15px;
}

.productoResultado:hover
{
        background-color: #EEEEEE;
}

.nombreProducto {
        width: 300px;
        float: left;
        cursor: pointer;
}

.precioProducto {
        width: 70px;
        font-weight: bold;
        text-align: right;
        float: left;
}

.agregarProducto
{
        width: 70px;
        font-weight: bold;
        cursor: pointer;
        color: Purple;
        float: right;
        text-align: right;
}

.agregarProducto:hover
{
        width: 20%;
        font-weight: bold;
        color: Red;
}

.detalleNombre
{
        font-size: 19px;
        font-weight: bold;
        color: Blue;
}

.detalleAgregar
{
        float: right;
}
```

```
.detalleDescripcion
{
       font-style: italic;
       color: Gray;
}

.detallePrecio
{
       font-size: 15px;
       padding: 15px;
}

.detalleCategoria
{
       color: red
}

.detalleFoto
{
       float: left;
       padding: 5px;
       top: 0px;
       max-height: 110px;
}


.filaCanasto
{
    background: #000099;
    margin: 1px;
    padding: 2px;
}

.totalCanasto
{
    background: #990000;
    font-weight: bold;
    font-size: large;
}

.canastoNombre
{
    width: 110px;
}

.canastoPrecio
{
    width: 40px;
    text-align: right
}
```

```
.canastoCantidad
{
    width: 70px;
    text-align: right
}

.canastoQuitar
{
    color: White;
    font-size:smaller;
    text-align: right;
    padding-left: 2px;
}

#btnFinalizar
{
    margin-top: 15px;
    font-size: larger;
    background-color: White;
}

#finalizacion
{
    width: 400px;
    background-color: White;
    border: 2px gray solid;
    padding: 15px;
}

#ventanaModal
{
     visibility: hidden;
     position: absolute;
     left: 0px;
     top: 0px;
     width:100%;
     height:100%;
     text-align:center;
     z-index: 1000;
     background-image:url(fondo_modal.gif);
}

#ventanaModal div
{
     margin: 100px auto;
}

body
{
    height: 100%;
}
```

index.js

```
window.onload = function() {
    $Ajax("categorias.aspx", {
        onfinish: cargarCategorias,
        tipoRespuesta: $tipo.JSON,
        onerror: function(e) { alert(e) }
    });

    $("btnBuscar").onclick = enviarBusqueda;
    $("listaCategorias").onchange = buscarCategoria;

    // Define el Droppable
    Droppables.add("listaCanasto", {
        onDrop: sueltaProducto,
        accept: ["productoResultado", "detalleFoto"]
    });
}


function cargarCategorias(lista) {
    // Itero entre cada categoría recibida
    for (var i=0; i<lista.length; i++) {
        var cat = lista[i];
        // Agrego la categoría a la lista de Opciones
        var opc = new Option(cat.nombre, cat.id);
        $("listaCategorias").options[$("listaCategorias").length] = opc;
    };

    // Cargo los productos de la primera categoría
    buscarCategoria();
}

function enviarBusqueda() {
    $Ajax("buscar.aspx?buscar=" + $F("textoBusqueda"), {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON,
        divCargando: "cargando"
    });
}

function buscarCategoria() {
    $Ajax("categoria.aspx?id=" + $F("listaCategorias"), {
        onfinish: cargarResultados,
        tipoRespuesta: $tipo.JSON,
        divCargando: "cargando"
    });
}

var listaResultados;
```

```
function cargarResultados(lista) {
    // Apago el cartel de Loading
    $(”cargando”).hide();

    // Guardo la lista en una variable global
    // para posible uso futuro
    listaResultados = lista;

    // Limpio la lista de resultados vieja
    $(”listaResultados”).innerHTML = ””;

    // Itero entre los resultados
    for (var i=0; i<lista.length; i++) {
        agregarResultado(lista[i]);
    }
}

var cacheProductos = [];

function agregarResultado(producto) {
    var div = document.createElement(”div”);
    div.className = ’productoResultado’;
    div.id = ’producto’ + producto.id;
    div.innerHTML += ” <div class=’nombreProducto’ onclick=’detalle(”
+ producto.id + ”)’ >” + producto.nombre + ”</div>”;
    div.innerHTML += ” <div class=’precioProducto’>$ ” +
producto.precio + ”</div>”;
    div.innerHTML += ” <div class=’agregarProducto’ onclick=’agre-
gar(” + producto.id + ”)’>Agregar</div>”;
    $(”listaResultados”).appendChild(div);
    // Agregamos el producto al hash
    cacheProductos[producto.id] = {
        nombre: producto.nombre,
        precio: producto.precio,
        // Definimos al producto como Draggable
        drag: new Draggable(”producto” + producto.id, {revert:true})
    }
}

function detalle(id) {
    // Tengo que ir a buscar al servidor los detalles del producto
    $Ajax(”producto.aspx?id=” + id, {
        onfinish: mostrarDetalle,
        tipoRespuesta: $tipo.JSON,
        avisoCargando: ”cargando”,
        cache: true
    });
}

function mostrarDetalle(producto) {
    // Creo el contenido de la zona de Detalle
```

```
    var html = "<div class='detalleNombre'>" + producto.nombre +
"</div>";
    if (producto.foto!= "") {
        // Si tiene foto, la incluimos
        html += "<img src='fotos/" + producto.foto + " ' class='deta-
lleFoto' "
             + "id='foto" + producto.id + " ' />";
    }
    html += "<input type='button' class='detalleAgregar' value='Agre-
gar al Canasto' "
            + "onclick='agregar(" + producto.id + ")' />";
    html += "<div class='detalleCategoria'>Categoría: " +
producto.categoria + "</div>";
    html += "<div class='detallePrecio'>Precio: $" + producto.precio+
"</div>";
    html += "<div class='detalleDescripcion'>q" + producto.descrip-
cion + "</div>";
    $("listaDetalle").innerHTML = html;

    new Draggable("foto" + producto.id, {revert:true});
}


// Vector global del canasto de compras
var canasto = [];

// Agrega un producto al canasto de compras
function agregar(id) {
    // Primero buscamos si el producto ya estaba en el canasto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
        } else {
            i++;
        }
    }

    if (encontrado) {
        // El producto ya estaba en el canasto, sólo aumentamos
        // su cantidad. "i" tiene la posición actual
        canasto[i].cantidad++;
    } else {
        // El producto no estaba en el canasto, lo agregamos
        // obtenemos los datos del producto desde el hash global
        var producto = cacheProductos[id];
        nuevoProducto = {
                         'id': id,
                         'nombre': producto.nombre,
                         'precio': producto.precio,
```

```
                        'cantidad': 1
                       };

        canasto[canasto.length] = nuevoProducto;
    }
    // Llama a la función que dibuja el canasto
    actualizarCanasto();
}

// Dibuja el canasto en la zona especificada
function actualizarCanasto()  {
    // Primero creamos la lista de productos
    var contenido = "";
    if (canasto.length==0) {
        contenido = "El canasto está vacío. <br />";
    } else {
        var filaCanasto;
        var total = 0;
        for (var i=0; i<canasto.length; i++) {
            // Creo una fila por cada producto
            filaCanasto = "<div class='filaCanasto' ";
            filaCanasto += " id='canasto_" + canasto[i].id + " '>";
            // Armamos cada fila con una tabla
            var tablaCanasto;
            tablaCanasto = "<table width='100%'><tr>";
            tablaCanasto += "<td class='canastoNombre'>" +
                  canasto[i].nombre + "</td>";
            tablaCanasto += "<td class='canastoCantidad'>  " + ca-
                  nasto[i].cantidad
                + "<img src='flechadown.gif' onclick='bajar(" + ca-
                  nasto[i].id + ")' />"
                + "<img src='flechaup.gif' onclick='subir(" + ca-
                  nasto[i].id + ")' /></td>";
            tablaCanasto += "<td class='canastoPrecio'>$" +
                  canasto[i].cantidad*canasto[i].precio + "</td>";
            tablaCanasto += "<td><a href='javascript:quitar(" + ca-
                  nasto[i].id
                + ")' class='canastoQuitar'>Quitar</a></td>";
            tablaCanasto += "</tr></table>";
            filaCanasto += tablaCanasto + "</div>";
            contenido += filaCanasto;
            total += canasto[i].precio * canasto[i].cantidad;
        }
        // Mostramos el TOTAL
        contenido += generarFilaTotal(total);

        // Agregamos el botón para Finalizar la Compra
        contenido += "<input type='button' id='btnFinalizar'
value='Finalizar Compra' />";
    }
    $("listaCanasto").innerHTML = contenido;
```

```
    // Agrego todas las filas como Draggables
    for (var i=0; i<canasto.length; i++) {
        new Draggable("canasto_" + canasto[i].id, {revert:true});
    }


    // Creamos el bote de basura
    var bote = document.createElement("img");
    bote.src = "basura.png";
    bote.width = "100";
    bote.height = "100";
    bote.id = "basura"
    $("listaCanasto").appendChild(bote);
    // Creo la zona de Drop con el bote
    Droppables.add("basura", {
        accept: 'filaCanasto',
        onDrop: tiraProducto
    });

    $("btnFinalizar").onclick = finalizar;
}

function generarFilaTotal(total) {
    var filaCanasto = "<div class='totalCanasto'> ";
    var tablaCanasto;
    tablaCanasto = "<table width='100%'><tr>";
    tablaCanasto += "<td class='canastoNombre'>TOTAL</td>";
    tablaCanasto += "<td class='canastoCantidad'></td>";
    tablaCanasto += "<td class='canastoPrecio'>$" + total+ "</td>";
    tablaCanasto += "<td><a href='javascript:vaciar()' class='canas-
                        toQuitar'>Vaciar</a></td>";
    tablaCanasto += "</tr></table>";
    filaCanasto += tablaCanasto + "</div>";
    return filaCanasto;
}

// Quita un producto del canasto
function quitar(id) {
    // Primero buscamos el producto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
            // Lo eliminamos de la lista con Prototype
            canasto = canasto.without(canasto[i]);
        } else {
            i++;
        }
    }
```

```
    actualizarCanasto();
}

// Vacía todo el canasto
function vaciar() {
    canasto = [];
    actualizarCanasto();
}

// Sube la cantidad de un producto del canasto
function subir(id) {
    // Primero buscamos el producto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
            // Lo eliminamos de la lista con Prototype
            canasto[i].cantidad++;
        } else {
            i++;
        }
    }
    actualizarCanasto();
}

// Baja la cantidad de un producto del canasto
function bajar(id) {
    // Primero buscamos el producto
    var i=0;
    var encontrado = false;
    while ((i<canasto.length) && (!encontrado)) {
        if (canasto[i].id == id) {
            encontrado = true;
            // Lo eliminamos de la lista con Prototype
            canasto[i].cantidad—;
            // Si quedó en cero, quitamos el producto
            if (canasto[i].cantidad==0) {
                quitar(id);
            }
        } else {
            i++;
        }
    }
    actualizarCanasto();
}

// Se ejecuta cuando el usuario suelta un producto en el canasto
function sueltaProducto(drag, drop, evento) {
    var id;
    if (drag.id.substring(0, 8)==”producto”) {
```

```
        // Obtenemos el ID, es productoXXX
        id = drag.id.substring(8, drag.id.length);
    } else {
        // Obtenemos el ID, es fotoXXX
        id = drag.id.substring(4, drag.id.length);
    }
    // Agregamos el producto al canasto
    agregar(id);
}

function tiraProducto(drag, drop, evento) {
    var id = drag.id.substring(8, drag.id.length);
    quitar(id);
}

// Muestra el formulario de Finalización de Compra
function finalizar() {
    $("ventanaModal").style.visibility="visible";
}

// Cancela el envío de la compra
function cancelar() {
    $("ventanaModal").style.visibility="hidden";
}

// Envía toda la información del pedido
function enviar() {
    // TODO: Validaciones
    // Armamos un string de tipo QueryString con los datos personales
    var parametros = "";
    parametros += "nomyape=" + $F("txtNombre");
    parametros += "&direccion=" + $F("txtDireccion");
    parametros += "&telefono=" + $F("txtTelefono");

    // Recorremos todo el canasto y armamos un string con los ID y
            las cantidades
    var strCarrito = "";
    for (var i=0; i<canasto.length; i++) {
        strCarrito += canasto[i].id;
        // Para separar ID de Cantidad usamos guion
        strCarrito += "-";
        strCarrito += canasto[i].cantidad;
        // Usamos punto para separar productos
        strCarrito += ".";
    }
    // Eliminamos último pipe
    strCarrito = strCarrito.substring(0, strCarrito.length-1);
    parametros += "&carrito=" + strCarrito;

    $Ajax("enviar.aspx", {
        metodo: $metodo.POST,
```

```
            parametros: parametros,
            avisoCargando: ”cargando”,
            onfinish: function() {
                alert(”Su pedido ha sido guardado”);
                cancelar();
                vaciar();
            }
        });
    }
```