



# **Sistemas Distribuidos e Internet**

## **Cliente para servicios Web con jQuery - Ajax**

### **Sesión- 10.3**

### **Curso 2017/ 2018**



## Aplicación cliente jQuery-Ajax

En este guión vamos a implementar una aplicación basada en jQuery-Ajax que definirá una interfaz de usuario y consumirá los servicios web anteriormente implementados.

Actualmente muchos sistemas web se dividen en dos o más aplicaciones, comúnmente una aplicación para la interfaz de usuario que consume los servicios de la lógica de negocio y una o varias aplicaciones que implementan los servicios de la lógica de negocio. La división en varias aplicaciones permite que estas puedan ser desplegadas en diferentes máquinas, favorece la escalabilidad de partes concretas de la aplicación y la orientación a los microservicios.

### Parte de los Servicios Web - Access-Control-Allow-Origin

Por defecto y por razones de seguridad la aplicación **TiendaMusica** no incluye las cabeceras **Access-Control-Allow-\*** en sus respuestas, si implementamos una aplicación cliente en nuestra misma máquina que haga peticiones contra esta aplicación nuestro navegador podría bloquearlas por motivos de seguridad.

Una solución durante la fase de desarrollo sería agregar en **app.js** las cabeceras más permisivas de Access-Control-Allow-Origin para todas las peticiones. Es muy importante agregar este **app.use** en primera posición.

```
// Módulos
var express = require('express');
var app = express();

app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Credentials", "true");
  res.header("Access-Control-Allow-Methods", "POST, GET, DELETE, UPDATE, PUT");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, token");
  // Debemos especificar todas las headers que se aceptan. Content-Type , token
  next();
});
```

### Aplicaciones Cliente - Single Page Application (SPA)

Como se comentó anteriormente se desarrollará aplicación que consumirá los servicios web de **tiendaMusica**. La aplicación va a ser de una única página (SPA), el propósito de estas aplicaciones es dar una experiencia de usuario más fluida, haciendo aparecer y desaparecer componentes de interfaz de usuario dinámicamente en lugar de recargar la página completa.

Existen muchas tecnologías que simplifican la creación de aplicaciones SPA, en este caso utilizaremos jQuery y AJAX, por ser las más populares y por poder ser usadas también para enriquecer interfaces de aplicaciones web de todo tipo (no hace falta que sean SPA).

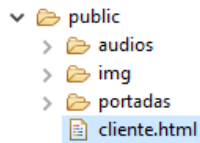
Descargamos el fichero **cliente.html** del campus virtual. Esta página será la base de un cliente jQuery que utilizará los servicios web REST de **tiendaMusica**.

Esta nueva aplicación podría ser desplegada dentro de una aplicación express o cualquier otro servidor web (basta con que admite HTML y JS, por ejemplo, un **Apache**).

En este caso para simplificar el desarrollo vamos a hacer que esta aplicación sea parte de nuestra aplicación anterior **tiendaMusica**. Copiamos **cliente.html** en el directorio **/public/** del



proyecto **tiendaMusica**. **Sí teníamos previamente algún fichero .html más en el directorio /public los eliminamos.**



Verificamos que contiene los scripts de **jquery** y **bootstrap** necesarios.

```
<title>jQuery uoMusic </title>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
```

**Ciente.html** va a ser la base de la aplicación, sobre este HTML cargaremos dinámicamente diferentes *componentes/widgets* (un widget con el formulario de identificación, otro para mostrar las canciones, etc.). Estos componentes se van a cargar sobre el <div> con id="contenedor-principal".

Agregamos un Script al final del fichero, incluimos las variables generales que van a ser utilizadas en todo el cliente (**token** y **URLbase**), aprovechamos también para cargar el **widget-login.html** sobre el contenedor principal (**widget-login** está aún sin implementar).

```
<!-- Contenido -->
<div class="container" id="contenedor-principal"> <!-- id para identificar -->

</div>

<script>

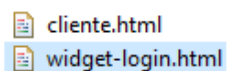
var token;
var URLbase = "http://localhost:8081/api";

$( "#contenedor-principal" ).load( "widget-login.html");

</script>
</body>
```

## Sistema de login

Creamos un nuevo fichero **widget-login.html** en la carpeta **/public/**.



Insertamos dos inputs, para el **email** y el **password**, y un **botón de envió**. En este caso el sistema de login va a ser ligeramente diferente al utilizado anteriormente.

1. No va a tener la etiqueta **<form>**, no se va a enviar por el método tradicional sino usando jQuery.



2. Para facilitar el acceso a los elementos de la página HTML desde jQuery es muy recomendable incluir **id** en todos los elementos que vayan a ser "manipulados".

```
<div id="widget-login">
  <div class="form-group">
    <label class="control-label col-sm-2" for="email">Email:</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" name="email"
        placeholder="email@email.com" id="email" />
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-sm-2" for="password">Password:</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" name="password"
        placeholder="contraseña" id="password"/>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="button" id="boton-login">Aceptar</button>
    </div>
  </div>
</div>
```

A continuación del código HTML implementaremos el script. En el script vamos a registrar la pulsación en el **botón-login**, como resultado se envía el contenido de los campos **email** y **password** al servicio POST <http://localhost:8081/ap/autenticar>.

Para realizar la petición usamos el objeto **\$.ajax**. Es importante declarar el **dataType** en la petición. La llamada a este servicio debería retornarnos un objeto JSON con una propiedad **token**, el cual debemos almacenar para futuras peticiones.

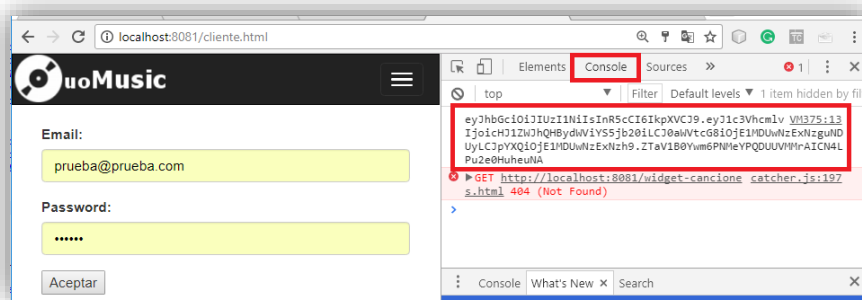
Una vez identificado con éxito vamos a mostrar automáticamente la lista de canciones del usuario. (Como los códigos de respuesta http del servicio están bien implementados la petición sabe gestionar los **success (200)** y los **error (401)**)

```
<script>
$( "#boton-login" ).click( function() {
  $.ajax({
    url: URLbase + "/autenticar",
    type: "POST",
    data: {
      email : $( "#email" ).val(),
      password : $( "#password" ).val()
    },
    dataType: 'json',
    success: function( respuesta ) {
      console.log( respuesta.token ); // <- Prueba
      token = respuesta.token;
      $( "#contenedor-principal" ).load( "widget-canciones.html" );
    },
    error : function (error){
      $( "#widget-login" )
        .prepend("<div class='alert alert-danger'>Usuario no encontrado</div>");
    }
  });
});
</script>
```



Es importante que el fragmento de `<script>` se añada después del código HTML, (de lo contrario estamos intentando registrar un evento `click()` en un botón **botón-login** que aún no ha incluido, otra opción para estar seguros de que los JS se ejecutan cuando toda la web está cargada es usar el `window.onload` o `$(document).ready`).

Accedemos a <http://localhost:8081/cliente.html> y comprobamos que nos devuelve el **token** (se muestra en la Consola (F12) ya que hemos incluido un mensaje `console.log(token)` una vez se identifica el usuario). El servicio debe estar activo para que el cliente funcione.



## Listar canciones

Ya disponemos del **token** de seguridad, podemos realizar una petición para obtener las canciones y listarlas. Creamos un documento **widget-canciones.html**

cliente.html  
widget-canciones.html  
widget-login.html

Comenzamos definiendo la vista HTML. Creamos una tabla con el cuerpo vacío, es importante identificar el cuerpo de la tabla con una **id** ya que será donde agregaremos las canciones.

```
<div id="widget-canciones" >
  <button class="btn" onclick="cargarCanciones()" >Actualizar</button>
  <table class="table table-hover">
    <thead>
      <tr>
        <th>Nombre</th>
        <th>Genero</th>
        <th>Precio</th>
        <th class="col-md-1"></th>
      </tr>
    </thead>
    <tbody id="tablaCuerpo">
    </tbody>
  </table>
</div>
```

A continuación de la vista HTML incluimos el script, definimos **cargarCanciones()** que hará una petición a **GET /cancion**, al obtener las canciones las insertará en el cuerpo de la tabla (**tablaCuerpo**).



Vamos a definir a una segunda función **actualizarTabla(canciones)** que se encargará de insertar las canciones en el HTML.

Invocamos a la función principal **cargarCanciones()** para que se carguen cuando se muestre el widget.

```
<script>
var canciones;

function cargarCanciones(){
    $.ajax({
        url: URLbase + "/cancion",
        type: "GET",
        data: { },
        dataType: 'json',
        headers: { "token": token },
        success: function(respuesta) {
            canciones = respuesta;
            actualizarTabla(canciones);
        },
        error : function (error){
            $( "#contenedor-principal" ).load("widget-login.html");
        }
    });
}

function actualizarTabla(cancionesMostrar){
    $( "#tablaCuerpo" ).empty(); // Vaciar la tabla
    for (i = 0; i < cancionesMostrar.length; i++) {
        $( "#tablaCuerpo" ).append(
            "<tr id="+cancionesMostrar[i]._id+">" +
            "<td>"+cancionesMostrar[i].nombre+"</td>" +
            "<td>"+cancionesMostrar[i].genero+"</td>" +
            "<td>"+cancionesMostrar[i].precio+"</td>" +
            "<td>" +
            "<a onclick=detalles('"+cancionesMostrar[i]._id+"')>Detalles</a><br>" +
            "<a onclick=eliminar('"+cancionesMostrar[i]._id+"')>Eliminar</a>" +
            "</td>" +
            "</tr>" );
        // Mucho cuidado con las comillas del eliminarCancion
        //la id tiene que ir entre comillas ' '
    }
}

cargarCanciones();
</script>
```

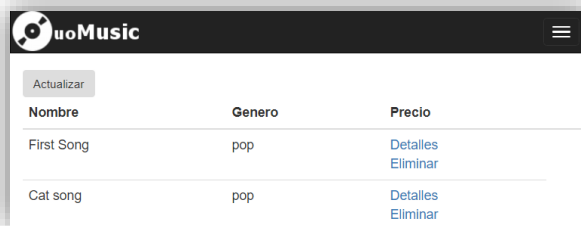
En cada registro de la tabla hemos incluido dos botones con enlaces a las funciones **detalles(id)** y **eliminar(id)**. Debemos de ser muy cuidadosos al hacer **appends** de código HTML que necesiten comillas. Es el caso del parámetro de las funciones **detalles(id)** y **eliminar(id)**. Si insertamos `onclick=(a3445a3)` en la página no funcionará, necesitamos `onclick=('a3445a3')`, la *id a3445a3 es un valor String, no el nombre de una variable.*

Comprobamos que la lista de canciones se muestra al identificarnos. El botón **Actualizar**, simplemente invoca de nuevo a la función **cargarCanciones()**.



Sí quisiéramos que las canciones se actualizaran de forma automática cada N segundos (de forma similar a los clientes de correo electrónico) podríamos incluir un **setInterval**.

```
setInterval(function() {  
    cargarCanciones();  
}, 5000);
```



En **cliente.html** implementamos la función **widgetCanciones()**, la cual se ejecuta al pulsar la opción del menú "Canciones". La utilizaremos para permitir que el usuario vuelva a la lista de canciones.

```
function widgetCanciones(){  
    $( "#contenedor-principal" ).load( "widget-canciones.html");  
}
```

## Eliminar canción

Implementamos la función **eliminar (\_id)** en **widget-canciones.html**. Si la petición de eliminar se completa con éxito (**success**), eliminamos el **<tr>** correspondiente de la canción (también podríamos volver a llamar a **cargarCanciones()** y recargar toda la tabla).

```
function eliminar( _id ) {  
    $.ajax({  
        url: URLbase + "/cancion/" + _id,  
        type: "DELETE",  
        data: { },  
        dataType: 'json',  
        headers: { "token": token },  
        success: function( respuesta ) {  
            console.log("Eliminada: " + _id);  
            $( "#"+_id ).remove(); // eliminar el <tr> de la canción  
        },  
        error : function (error){  
            $( "#contenedor-principal" ).load("widget-login.html");  
        }  
    });  
}
```

Comprobamos que la acción eliminar funciona de forma correcta.



## Ver detalles de canción

Vamos a implementar la función **detalles(id)**, correspondiente a **widget-canciones.html**, esta función se invoca al pulsar en el enlace detalles de cada canción. **detalles(id)** debe:

1. Guardar en la variable global **idCancionSeleccionada** la canción seleccionada. Esta variable va a utilizarse para intercambiar datos entre widgets, anteriormente habíamos hecho algo similar con la variable **token**. (No es estrictamente necesario declarar la variable **idCancionSeleccionada** en ningún sitio, aunque si podríamos declararla en **cliente.html**)
2. Cargar la vista **widget-detalles.html**, la cual implementaremos posteriormente.

```
cargarCanciones();  
  
function detalles(_id) {  
    idCancionSeleccionada = _id;  
    $( "#contenedor-principal" ).load( "widget-detalles.html");  
}
```

Para mostrar los detalles de la crearemos el fichero **widget-detalles.html**

cliente.html  
widget-canciones.html  
widget-detalles.html  
widget-login.html

Como en el caso anterior incluimos primero la vista en HTML, es importante incluir **ids** en los elementos HTML a los que debemos acceder posteriormente.

```
<div id="widget-detalles">  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="detalles-nombre">Nombre:</label>  
    <div class="col-sm-10">  
      <input type="text" class="form-control" name="detalles-nombre"  
        placeholder="Nombre de mi canción" id="detalles-nombre" readonly/>  
    </div>  
  </div>  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="detalles-genero">Genero:</label>  
    <div class="col-sm-10">  
      <input type="text" class="form-control" name="detalles-genero"  
        placeholder="Nombre de mi canción" id="detalles-genero" readonly/>  
    </div>  
  </div>  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="detalles-precio">Precio (€):</label>  
    <div class="col-sm-10">  
      <input type="number" step="0.01" class="form-control" name="detalles-precio"  
        placeholder="2.50" id="detalles-precio" readonly/>  
    </div>  
  </div>  
  <button onclick="widgetCanciones()" class="btn" >Volver</button>  
</div>
```

A continuación, vamos a incluir en **widget-detalles.html** el script, supondremos que la variable **idCancionSeleccionada** contiene la **id** de la canción que queremos cargar, realizamos la petición para obtener la canción y después cargamos los valores en los inputs.





```
<script>

$.ajax({
  url : URLbase + "/cancion/" + idCancionSeleccionada ,
  type : "GET",
  data : {},
  dataType : 'json',
  headers : {
    "token" : token
  },
  success : function(cancion) {
    $("#detalles-nombre").val(cancion.nombre);
    $("#detalles-genero").val(cancion.genero);
    $("#detalles-precio").val(cancion.precio);
  },
  error : function(error) {
    $( "#contenedor-principal" ).load("widget-login.html");
  }
});

</script>
```

Ejecutamos la aplicación y comprobamos que muestra los detalles de la canción correctamente.

### Agregar canción

Abrimos el fichero **widget-canciones.html** y agregamos un nuevo botón en su vista HTML.

```
<div id="widget-canciones" >
  <button class="btn btn-primary" onclick="widgetAgregar()">Nueva Canción</button>
  <button class="btn" onclick="cargarCanciones()" >Actualizar</button>
  <table class="table table-hover">
```

Este botón va a ejecutar la función **widgetAgregar()**, la cual nos dirigirá al widget que muestra el formulario para agregar una nueva canción.

```
function widgetAgregar() {
  $( "#contenedor-principal" ).load( "widget-agregar.html");
}
```

Creamos el fichero **widget-agregar.html** en la carpeta **/public**.

- cliente.html
- widget-agregar.html
- widget-canciones.html
- widget-detalles.html
- widget-login.html

Todos los inputs a los que tenemos que acceder están identificados con una **id** (Hay que recordar que las id deben ser únicas, no puede haber dos elementos "nombre" o "genero"). Al igual que el botón de agregar canción, el cual ejecuta la función **agregarCancion()** (que está aún sin implementar).

```
<div id="widget-agregar" >
  <div class="form-group">
    <label class="control-label col-sm-2" for="agregar-nombre">Nombre:</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" name="agregar-nombre">
```



```
placeholder="Nombre de mi canción" id="agregar-nombre" />
</div>
</div>
<div class="form-group">
<label class="control-label col-sm-2" for="agregar-genero">Genero:</label>
<div class="col-sm-10">
<input type="text" class="form-control" name="agregar-genero"
placeholder="Nombre de mi canción" id="agregar-genero" />
</div>
</div>
<div class="form-group">
<label class="control-label col-sm-2" for="agregar-precio">Precio (€):</label>
<div class="col-sm-10">
<input type="number" step="0.01" class="form-control" name="detalles-precio"
placeholder="2.50" id="agregar-precio" />
</div>
</div>
<div class="col-sm-offset-2 col-sm-10">
<button type="button" class="btn btn-primary" id="boton-agregar"
onclick="agregarCancion()">Agregar</button>
</div>
</div>
```

Implementamos la función **agregarCanción()** dentro de **widget-agregar.html**, tendrá la siguiente funcionalidad:

1. Obtiene los datos de la canción de los inputs declarados en la vista HTML utilizando sus IDs, después realiza una petición **POST** a **http://localhost:8081/api/cancion**
2. Una vez agregada vuelve a mostrar la lista de las canciones

```
<script>
function agregarCancion( ) {
$.ajax({
url: URLbase + "/cancion",
type: "POST",
data: {
nombre : $("#agregar-nombre").val(),
genero : $("#agregar-genero").val(),
precio : $("#agregar-precio").val()
},
dataType: 'json',
headers: { "token": token },
success: function(respuesta) {
console.log(respuesta); // <-- Prueba
$("#contenedor-principal").load("widget-canciones.html");
},
error : function (error){
$("#contenedor-principal").load("widget-login.html");
}
});
}
</script>
```

Ejecutamos la aplicación y comprobamos que nos permite agregar canciones.



Nombre	Genero	Precio
First Song	pop	<a href="#">Detalles</a> <a href="#">Eliminar</a>
Cat song	pop	<a href="#">Detalles</a> <a href="#">Eliminar</a>
Sound sea	pop	<a href="#">Detalles</a> <a href="#">Eliminar</a>

## (Ampliación) Filtrado dinámico

Implementaremos un sistema de filtrado en la lista de canciones **widget-canciones.html**, de forma que se muestren solo las canciones que coincidan con una cadena de texto.

Abrimos el fichero **widget-canciones.html** incluimos un input en la parte superior con la id **filtro-nombre** (para poder referenciarlo desde el código JS).

```
<div id="widget-canciones" >  
  <input type="text" class="form-control" placeholder="Filtrar por nombre"  
  id="filtro-nombre"/>  
  <button class="btn btn-primary" onclick="widgetAgregar()">Nueva Cancion</button>  
  <button class="btn" onclick="cargarCanciones()">Actualizar</button>  
</div>
```

En la parte del Script implementamos un escuchador sobre el input con id **filtro-nombre**, cada vez que se detecte un cambio sobre él recorreremos la lista de canciones originales y guardamos en el array **cancionesFiltradas** las canciones que tienen coincidencia con el texto introducido, finalmente representamos las canciones contenidas en el array **cancionesFiltradas**.

```
$('#filtro-nombre').on('input',function(e){  
  var cancionesFiltradas = [];  
  var nombreFiltro = $('#filtro-nombre').val();  
  
  for (i = 0; i < canciones.length; i++) {  
    if (canciones[i].nombre.indexOf(nombreFiltro) != -1 ){  
      cancionesFiltradas.push(canciones[i]);  
    }  
  }  
  actualizarTabla(cancionesFiltradas);  
});
```

Nombre	Genero	Precio
Cuatro	pop	1



## (Ampliación) Ordenación

En el propio **widget-canciones.html** incluiremos un sistema de ordenación por **precio** y **nombre**. Incluimos enlaces en la cabecera de la tabla para Nombre y Precio que invocarán respectivamente a **ordenarPorNombre()** y **ordenarPorPrecio()**.

```
<thead>
  <tr>
    <th><a onclick="ordenarPorNombre()">Nombre</a></th>
    <th>Genero</th>
    <th><a onclick="ordenarPorPrecio()">Precio</a></th>
    <th class="col-md-1"></th>
  </tr>
</thead>
```

Implementamos la función **ordenarPorPrecio()**, para ordenar el array utilizamos la función **sort**, esta función espera recibir dos canciones **a** y **b**. Dependiendo del retorno de la función de ordenación colocará una canción delante de otra.

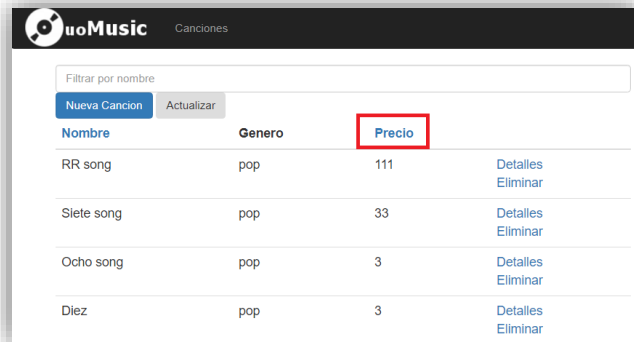
- $>0$  la canción **a** tiene más prioridad
- $<0$  la canción **b** tiene más prioridad
- $=0$  las canciones tienen la misma prioridad

Por el momento ordenados el precio de mayor a menor y los nombres de menor a mayor ( **A < B** )

```
function ordenarPorPrecio() {
  canciones.sort(function(a, b) {
    if (parseFloat(a.precio) > parseFloat(b.precio)) return -1;
    if (parseFloat(a.precio) < parseFloat(b.precio)) return 1;
    return 0;
    // Dependiendo de retorno >0 =0 o <0
  });
  actualizarTabla(canciones);
}

function ordenarPorNombre() {
  canciones.sort(function(a, b) {
    if (a.nombre > b.nombre) return 1;
    if (a.nombre < b.nombre) return -1;
    return 0;
  });
  actualizarTabla(canciones);
}
```

Sí ejecutamos la aplicación y pulsamos sobre las cabeceras **Nombre** o **Precio** de la tabla modificaremos el orden por defecto.



Nombre	Genero	Precio	
RR song	pop	111	<a href="#">Detalles</a> <a href="#">Eliminar</a>
Siete song	pop	33	<a href="#">Detalles</a> <a href="#">Eliminar</a>
Ocho song	pop	3	<a href="#">Detalles</a> <a href="#">Eliminar</a>
Diez	pop	3	<a href="#">Detalles</a> <a href="#">Eliminar</a>

Sí quisiésemos que con cada click sobre el enlace se invirtiera el orden de ordenación sería sencillo de implementar, bastaría con guardar un boolean con el estado de cada orden y en función del estado elegir un orden u otro y finalmente invertir el estado. Para implementar los dos ordenes en el campo precio haríamos lo siguiente:

```
var precioDsc = true;

function ordenarPorPrecio() {
  if (precioDsc) {
    canciones.sort(function(a, b) {
      return parseFloat(a.precio) - parseFloat(b.precio);
    });
  } else {
    canciones.sort(function(a, b) {
      return parseFloat(b.precio) - parseFloat(a.precio);
    });
  }
  actualizarTabla(canciones);
  precioDsc = !precioDsc; //invertir
}
```

### (Ampliación) Cookies

Podríamos almacenar el **token** en una Cookie para que la información no se pierda al refrescar el navegador. Ahora mismo cada vez que hacemos una actualización del navegador nos vuelve al **login**, lo cual es lógico ya que se vuelve a cargar toda la aplicación JavaScript.

JS contiene un motor de Cookies nativo, pero en este caso usaremos una librería <https://github.com/js-cookie/js-cookie> que simplifica notablemente su uso. Descargamos el fichero **cookie.js** del campus virtual y lo copiamos en la carpeta **/public/js/**.



Agregamos el fichero **cookie.js** a el fichero principal **cliente.html**.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```



```
<script src="/js/cookie.js"></script>
```

Abrimos el script de código del fichero **widget-login.html**, al recibir el token lo guardamos en una cookie bajo la clave **token**. Al identificarse incorrectamente debemos eliminar la cookie **token**.

```
success: function(respuesta) {  
    console.log(respuesta.token); // <- Prueba  
    token = respuesta.token;  
    Cookies.set('token', respuesta.token);  
    $('#contenedor-principal').load( "widget-canciones.html" );  
},  
error : function (error){  
    Cookies.remove('token');  
    $('#widget-login' )  
    .prepend("<div class='alert alert-danger'>Usuario no encontrado</div>");  
}
```

En el script de código del fichero principal **cliente.html** comprobaremos si hay una cookie **token** activa, si es así guardamos la cookie en la variable **token** (como si ya hubiésemos pasado por el login) e intentamos cargar el **widget-canciones**.

```
var token;  
var URLbase = "http://localhost:8081/api";  
  
$( "#contenedor-principal" ).load( "widget-login.html");  
  
if ( Cookies.get('token') != null ){  
    token = Cookies.get('token');  
    $( "#contenedor-principal" ).load( "widget-canciones.html");  
}
```

A partir de este momento si nos hemos identificado y refrescamos la página no debería pedirnos identificarnos de nuevo.

### (Ampliación) Rutas

Toda la aplicación se encuentra sobre la ruta <http://localhost:8081/cliente.html>, esto imposibilita la compartición de URLs a partes concretas de la aplicación. Aunque esto puede no ser un problema en algunas aplicaciones en algunos casos sí que se podrían requerir esta funcionalidad.

Una solución podría ser implementar un sistema de URLs utilizando parámetros GET. En este caso vamos a implementar acceso a dos de las partes de la aplicación: **cliente.html?w=login** y **cliente.html?w=canciones** (podríamos decidir incluir más URLs).

Al inicio del script de **widget-login.html** incluimos la sentencia que modifica la URL actual del navegador, agregando **?w=login**

```
<script>  
window.history.pushState("", "", "/cliente.html?w=login");
```



Repetimos el proceso en **widget-canciones.html** , agregando **?w=canciones**

```
<script>
window.history.pushState("", "", "/cliente.html?w=canciones");
```

De esta forma el parámetro **w** se va a agregar a la URL cuando el cliente abra las vistas.

Solo nos falta que **cliente.html** sea capaz de cargar los componentes adecuados en función del parámetro **w**. (utilizaremos la función **searchParams** para obtener el parámetro, esta función puede no funcionar en navegadores antiguos).

```
<script>

var token;
var URLbase = "http://localhost:8081/api";

$( "#contenedor-principal" ).load( "widget-login.html" );

if ( Cookies.get('token') != null ){
    token = Cookies.get('token');
    $( "#contenedor-principal" ).load( "widget-canciones.html" );

    var url = new URL(window.location.href);
    var w = url.searchParams.get("w");
    switch(w) {
        case "login":
            $( "#contenedor-principal" ).load("widget-login.html");
            break;
        case "canciones":
            $( "#contenedor-principal" ).load("widget-canciones.html");
            break;
        default:
            $( "#contenedor-principal" ).load( "widget-canciones.html" );
    }
}
```

Probamos el funcionamiento de las URLs:

<http://localhost:8081/cliente.html?w=login> y

<http://localhost:8081/cliente.html?w=canciones>

Posibles mejoras

- En cada widget separar el código HTML del JS.
- Realizar comprobaciones en los parámetros introducidos por los usuarios, incluyendo mensajes que adviertan al usuario de los errores.
- Deshabilitar o modificar la funcionalidad del botón “back” del navegador.