Programación Web
Tema 3.3 AJAX

Miguel Ángel Manso
Emerson Castañeda
Ramón Alcarria
ETSI en Topografía, Geodesia y Cartografía - UPM

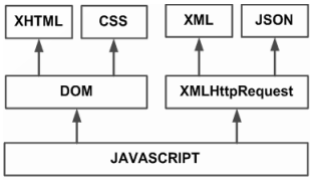
Introducción (I)

- **AJAX** aparece por primera vez en el artículo de Jesse James Garrett el 18 de Febrero de 2005, titulado : *"Ajax: A New Approach to Web Applications"*
- **AJAX** = **Asynchronous JavaScript + XML** (JavaScript asíncrono + XML).
- El artículo define AJAX como:

  "Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes."

Introducción (II)

- Las tecnologías que forman **AJAX** son:
  - **XHTML** y **CSS**, para crear una presentación basada en estándares
  - **DOM**, para interactuar y manipular dinámicamente la presentación
  - **XML**, **XSLT** y **JSON**, para intercambiar y manipular la información
  - **XMLHttpRequest**, para el intercambio asíncrono de información
  - **JavaScript**, para unir todas las demás tecnologías



Desarrollar aplicaciones AJAX requiere un conocimiento avanzado de cada una de éstas tecnologías

Introducción (II)

Ejercicio 0

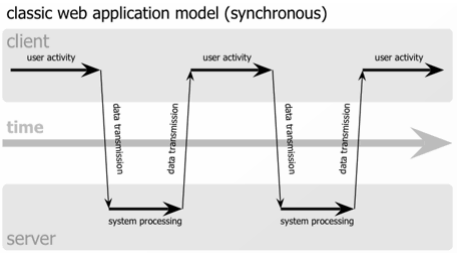
- Representar este JSON en una tabla

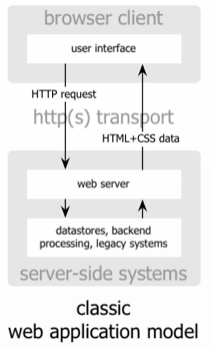
[{ "Name" : "Alfreds Futterkiste", "City" : "Berlin", "Country" : "Germany" }, { "Name" : "Berglunds snabbköp", "City" : "Luleå", "Country" : "Sweden" }, { "Name" : "Centro comercial Moctezuma", "City" : "México D.F.", "Country" : "Mexico" }, { "Name" : "Ernst Handel", "City" : "Graz", "Country" : "Austria" }, { "Name" : "FISSA Fabrica Inter. Salchichas S.A.", "City" : "Madrid", "Country" : "Spain" }, { "Name" : "Galería del gastrónomo", "City" : "Barcelona", "Country" : "Spain" }, { "Name" : "Island Trading", "City" : "Cowes", "Country" : "UK" }, { "Name" : "Königlich Essen", "City" : "Brandenburg", "Country" : "Germany" }, { "Name" : "Laughing Bacchus Wine Cellars", "City" : "Vancouver", "Country" : "Canada" }, { "Name" : "Magazzini Alimentari Riuniti", "City" : "Bergamo", "Country" : "Italy" }, { "Name" : "North/South", "City" : "London", "Country" : "UK" }, { "Name" : "Paris spécialités", "City" : "Paris", "Country" : "France" }, { "Name" : "Rattlesnake Canyon Grocery", "City" : "Albuquerque", "Country" : "USA" }, { "Name" : "Simons bistro", "City" : "København", "Country" : "Denmark" }, { "Name" : "The Big Cheese", "City" : "Portland", "Country" : "USA" }, { "Name" : "Vaffeljernet", "City" : "Århus", "Country" : "Denmark" }, { "Name" : "Wolski Zajazd", "City" : "Warszawa", "Country" : "Poland" }]

Introducción (III)

Tradicionalmente, en las aplicaciones web, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan las llamadas al servidor.

Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

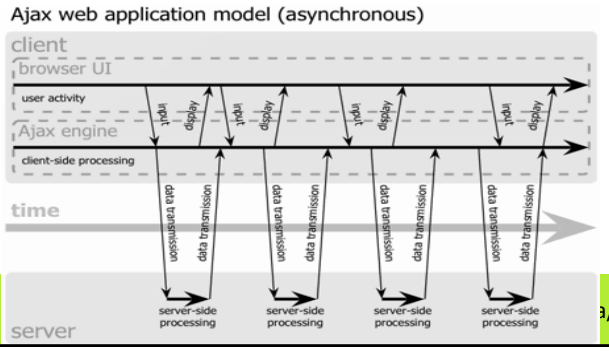


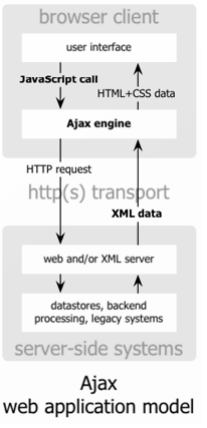


Introducción (IV)

Las aplicaciones basadas en AJAX evitan tener que recargar toda la página, creando un elemento intermedio entre el usuario y el servidor.

La capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.



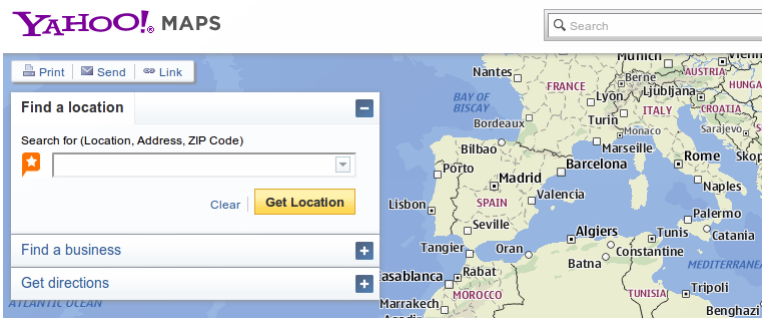


Introducción (V)

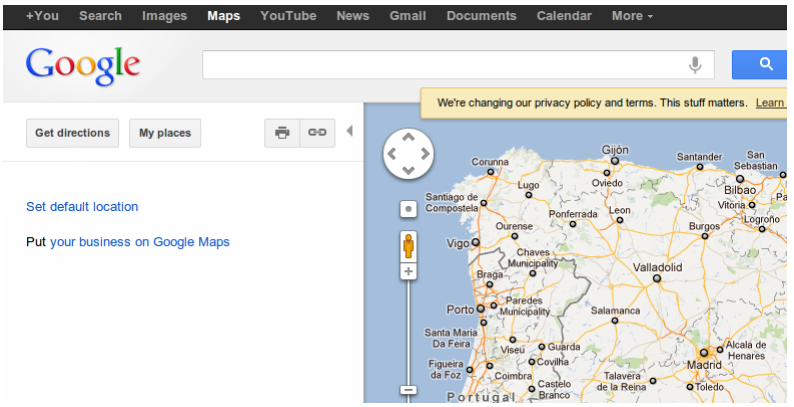
- Las peticiones **HTTP** al servidor se sustituyen por peticiones **JavaScript** que se realizan al elemento encargado de **AJAX**. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata.
- Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante **AJAX**. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Introducción (VI)

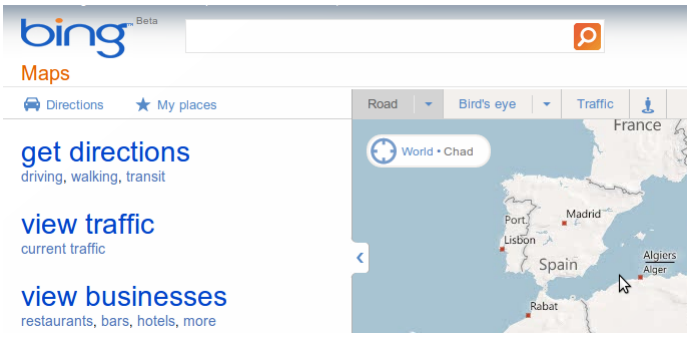
- Algunas aplicaciones, conocidas y basadas en **AJAX**:



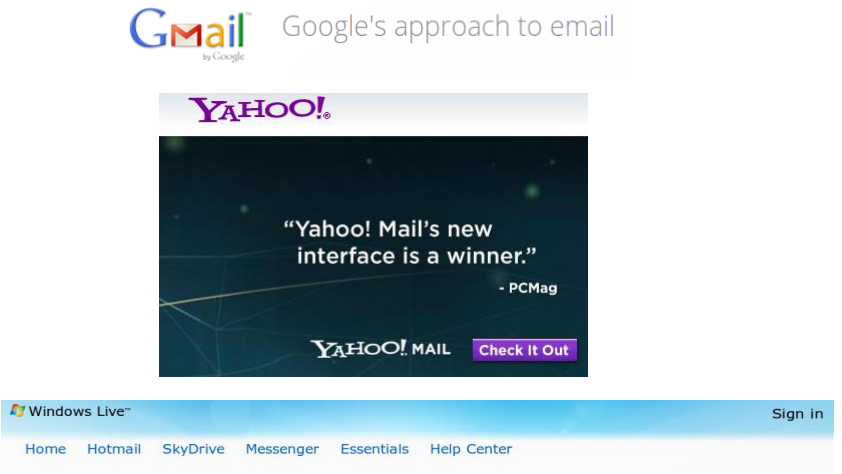
Introducción (VII)



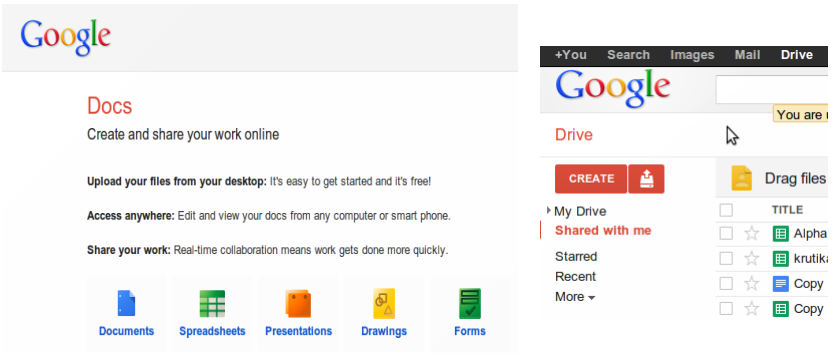
Introducción (VIII)



Introducción (IX)



Introducción (X)



Primeros Pasos (IV)

- **La primera aplicación**

La aplicación **AJAX** completa más sencilla consiste en una adaptación del clásico **"Hola Mundo"**

En este caso, una aplicación **JavaScript** descarga un archivo del servidor y muestra su contenido sin necesidad de recargar la página

Cuando se carga la página se ejecuta el método **JavaScript** que muestra el contenido de un archivo llamado **holamundo.txt** que se encuentra en el servidor.

Lo relevante es que la petición **HTTP** y la descarga de los contenidos del archivo se realizan sin necesidad de recargar la página

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title>Hola Mundo con AJAX</title>
        <script type="text/javascript">
            function descargaArchivo() {
                // Obtener la instancia del objeto XMLHttpRequest
                if(window.XMLHttpRequest) {
                    peticion_http = new XMLHttpRequest();
                }
                else if(window.ActiveXObject) {
                    peticion_http =
                    new ActiveXObject("Microsoft.XMLHTTP");
                }
                // Preparar la funcion de respuesta
                peticion_http.onreadystatechange = muestraContenido;
                // Realizar peticion HTTP
                peticion_http.open('GET',
                'http://localhost/holamundo.txt', true);
                peticion_http.send(null);
                function muestraContenido() {
                    if(peticion_http.readyState == 4) {
                        if(peticion_http.status == 200) {
                            alert(peticion_http.responseText);
                        }
                    }
                }
            }
            window.onload = descargaArchivo;
        </script>
    </head>
    <body>
    </body>
</html>
```


Primeros Pasos (V)

- La aplicación **AJAX** del ejemplo anterior se compone de cuatro grandes bloques:
  - Instanciar el objeto **XMLHttpRequest**
  - Preparar la función de respuesta
  - Realizar la petición al servidor
  - Ejecutar la función de respuesta

Preparar la función de respuesta

- Una vez obtenida la instancia del objeto **XMLHttpRequest**, se prepara la función que se encarga de procesar la respuesta del servidor. La propiedad ***onreadystatechange*** del objeto **XMLHttpRequest** permite indicar esta función directamente incluyendo su código mediante una función anónima o indicando una referencia a una función independiente

- El código anterior indica que cuando la aplicación reciba la respuesta del servidor, se debe ejecutar la función ***muestraContenido()***

Realizar la petición al servidor

- Después de preparar la aplicación para la respuesta del servidor, se realiza la petición **HTTP** al servidor:
  - peticion_http.open('GET', 'http://localhost/prueba.txt', true);
  - peticion_http.send(null);

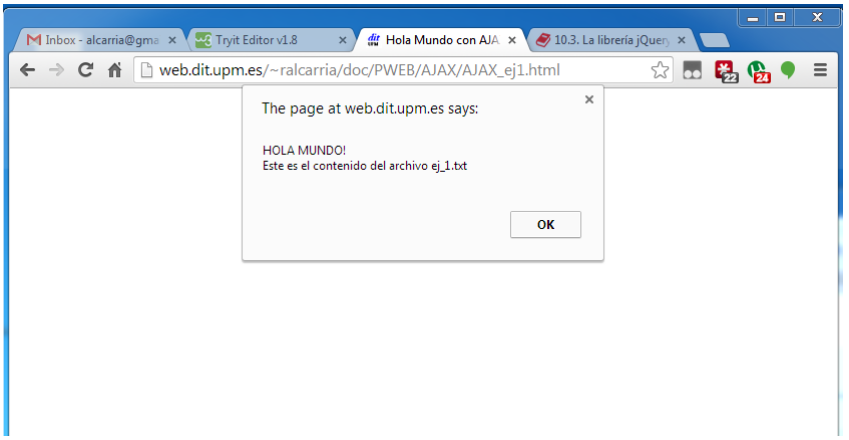
- Las instrucciones anteriores realizan el tipo de petición más sencillo que se puede enviar al servidor. En concreto, se trata de una petición de tipo **GET** simple que no envía ningún parámetro al servidor. La petición **HTTP** se crea mediante el método open(), en el que se incluye el tipo de petición **(GET)**, la **URL** solicitada (http://localhost/prueba.txt) y un tercer parámetro que vale **true**

Ejecutar la función de respuesta

- Una vez creada la petición **HTTP**, se envía al servidor mediante el método **send()**. Este método incluye un parámetro que en el ejemplo anterior vale null
- Por último, cuando se recibe la respuesta del servidor, la aplicación ejecuta de forma automática la función establecida anteriormente
- La función ***muestraContenido()*** comprueba en primer lugar que se ha recibido la respuesta del servidor (mediante el valor de la propiedad ***readyState***). Si se ha recibido alguna respuesta, se comprueba que sea válida y correcta (comprobando si el código de estado ***HTTP*** devuelto es igual a 200). Una vez realizadas las comprobaciones, simplemente se muestra por pantalla el contenido de la respuesta del servidor (en este caso, el contenido del archivo solicitado) mediante la propiedad ***responseText***

Ejecutar la función de respuesta

- Ver Ejercicio 1



Métodos y propiedades del objeto XMLHttpRequest

- El objeto **XMLHttpRequest** posee muchas otras propiedades y métodos diferentes a las manejadas por la primera aplicación de **AJAX**. A continuación se muestra la lista completa de todas las propiedades y métodos del objeto y todos los valores numéricos de sus propiedades

Propiedad	Descripción
`readyState`	Valor numérico (entero) que almacena el estado de la petición
`responseText`	El contenido de la respuesta del servidor en forma de cadena de texto
`responseXML`	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM
`status`	El código de estado HTTP devuelto por el servidor (`200` para una respuesta correcta, `404` para "No encontrado", `500` para un error de servidor, etc.)
`statusText`	El código de estado HTTP devuelto por el servidor en forma de cadena de texto: "OK", "Not Found", "Internal Server Error", etc.

Primeros Pasos (XI)

- Los valores definidos para la propiedad **readyState** son los siguientes:

Valor	Descripción
`0`	No inicializado (objeto creado, pero no se ha invocado el método `open`)
`1`	Cargando (objeto creado, pero no se ha invocado el método `send`)
`2`	Cargado (se ha invocado el método `send`, pero el servidor aún no ha respondido)
`3`	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad `responseText`)
`4`	Completo (se han recibido todos los datos de la respuesta del servidor)

Primeros Pasos (XII)

- Los métodos disponibles para el objeto **XMLHttpRequest** son los siguientes:

Método	Descripción
`abort()`	Detiene la petición actual
`getAllResponseHeaders()`	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor
`getResponseHeader("cabecera")`	Devuelve una cadena de texto con el contenido de la cabecera solicitada
`onreadystatechange`	Responsable de manejar los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función JavaScript
`open("metodo", "url")`	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino (puede indicarse de forma absoluta o relativa)
`send(contenido)`	Realiza la petición HTTP al servidor
`setRequestHeader("cabecera", "valor")`	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método `open()` antes que `setRequestHeader()`

Primeros Pasos (XIII)

- El método **open()** requiere dos parámetros (método HTTP y URL) y acepta de forma opcional otros tres parámetros
- ***open(string metodo, string URL [,boolean asincrono, string usuario, string password]);***
- Por defecto, las peticiones realizadas son asíncronas. Si se indica un valor **false** al tercer parámetro, la petición se realiza de forma síncrona, esto es, se detiene la ejecución de la aplicación hasta que se recibe de forma completa la respuesta del servidor
- No obstante, las peticiones síncronas son justamente contrarias a la filosofía de **AJAX**. El motivo es que una petición síncrona congela el navegador y no permite al usuario realizar ninguna acción hasta que no se haya recibido la respuesta completa del servidor. La sensación que provoca es que el navegador se ha colgado por lo que no se recomienda el uso de peticiones síncronas

Primeros Pasos (XIV)

- Los últimos dos parámetros opcionales permiten indicar un nombre de usuario y una contraseña válidos para acceder al recurso solicitado

- Por otra parte, el método **send()** requiere de un parámetro que indica la información que se va a enviar al servidor junto con la petición **HTTP**. Si no se envían datos, se debe indicar un valor igual a **null**. En otro caso, se puede indicar como parámetro una cadena de texto, un array de bytes o un objeto **XML DOM**

http://www.w3schools.com/ajax/ajax_aspphp.asp

Primeros Pasos (XXVI-b)

- ***encodeURIComponent()*** reemplaza todos los caracteres que no se pueden utilizar de forma directa en las **URL** por su representación hexadecimal. Las letras, números y los caracteres - _ . ! ~ * ' ( ) no se modifican

- Las sustituciones más conocidas son las de los espacios en blanco por ***%20***, y la del símbolo ***&*** por ***%26***. También se sustituyen todos los acentos y cualquier otro carácter que no se puede incluir directamente en una **URL**

Primeros Pasos (XXVII)

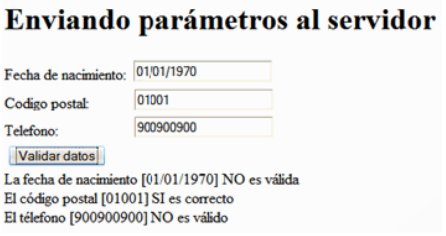
- JavaScript incluye una función contraria llamada ***decodeURIComponent()*** y que realiza la transformación inversa. Además, también existen las funciones ***encodeURI()*** y ***decodeURI()*** que codifican/decodifican una URL completa. La principal diferencia entre ***encodeURIComponent()*** y ***encodeURI()*** es que esta última no codifica los caracteres ***; / ? : @ & = + $, #***

```
var cadena = "http://www.ejemplo.com/ruta1/index.php?parametro=valor con ñ y &";

var cadena_segura = encodeURIComponent(cadena);
// cadena_segura = "http%3A%2F%2Fwww.ejemplo.com%2Fruta1%2Findex.php%3Fparametro%3Dvalor%20con%20%C3%B1%20y%20%26";

var cadena_segura = encodeURI(cadena);
// cadena_segura = "http://www.ejemplo.com/ruta1/index.php?parametro=valor%20con%20%C3%B1%20y%20";
```


Primeros Pasos (XXIX)



- En las aplicaciones reales, las validaciones de datos mediante *AJAX* sólo se utilizan en el caso de validaciones complejas que no se pueden realizar mediante el uso de código **JavaScript** básico
- En general, las validaciones complejas requieren el uso de bases de datos: comprobar que un nombre de usuario no esté previamente registrado, comprobar que la localidad se corresponde con el código postal indicado, etc

Primeros Pasos (XXIX)

- Ver ejercicio 2

Primeros Pasos (XXXIX)

```
{
mensaje: "...",
parametros: {telefono: "...", codigo_postal: "...", fecha_nacimiento: "..." }
}
```

- La función que se encarga de procesar la respuesta del servidor

```
function procesaRespuesta() {
  if(http_request.readyState == READY_STATE_COMPLETE) {
    if(http_request.status == 200) {
      var respuesta_json = http_request.responseText;
      var objeto_json = eval("("+respuesta_json+")");
      var mensaje = objeto_json.mensaje;
      var telefono = objeto_json.parametros.telefono;
      var fecha_nacimiento = objeto_json.parametros.fecha_nacimiento;
      var codigo_postal = objeto_json.parametros.codigo_postal;
      document.getElementById("respuesta").innerHTML = mensaje + "<br>"
      + "Fecha nacimiento = " + fecha_nacimiento + "<br>" +
      "Codigo postal = " + codigo_postal + "<br>" + "Telefono = "
      + telefono;
    }
  }
}
```

Primeros Pasos (XL)

- La respuesta **JSON** del servidor se obtiene mediante la propiedad ***responseText***

```
var respuesta_json = http_request.responseText;
```

- Sin embargo, esta propiedad solamente devuelve la respuesta del servidor en forma de cadena de texto. Para trabajar con el código **JSON** devuelto, se debe transformar esa cadena de texto en un objeto **JSON**.
- La forma más sencilla de realizar esa conversión es mediante la función **eval()**, en la que deben añadirse paréntesis al principio y al final para realizar la evaluación de forma correcta

```
var objeto_json = eval("("+respuesta_json+")");
```


Primeros Pasos (XLI)

- Una vez realizada la transformación, el objeto **JSON** ya permite acceder a sus métodos y propiedades mediante la notación de puntos tradicional. Comparado con las respuestas **XML**, este procedimiento permite acceder a la información devuelta por el servidor de forma mucho más simple

```
// Con JSON
var fecha_nacimiento = objeto_json.parametros.fecha_nacimiento;
// Con XML
var parametros = root.getElementsByTagName("parametros")[0];
var fecha_nacimiento =
parametros.getElementsByTagName("fecha_nacimiento")[0].firstChild.nodeValue;
```

Ver EJ 3

La respuesta del servidor es un objeto JSON con la siguiente estructura:

El nombre de usuario está libre:

```
{ disponible: "si" }
```

El nombre de usuario está ocupado:

```
{ disponible: "no", alternativas: ["...", "...", ..., "..."] }
```

Aplicaciones

Listas desplegables encadenadas

- Algunas aplicaciones web disponen de varias listas desplegables encadenadas. En este tipo de listas, cuando se selecciona un elemento de la primera lista desplegable, se cargan en la segunda lista unos valores que dependen del valor seleccionado en la primera lista
- Una posible implementación consiste en actualizar las listas desplegables mediante **AJAX**
- Los valores de la primera lista se incluyen en la página web y cuando se selecciona un valor de la lista, se realiza una consulta al servidor que devuelve los valores que se deben mostrar en la otra lista desplegable

Listas desplegables encadenadas

- Ver EJ 4

1) El script del servidor utilizado para cargar las provincias se llama `cargaProvinciasJSON.php` y la respuesta del servidor tiene el siguiente formato:

```
{ "01": "Álava/Araba",  "02": "Albacete",  "03": "Alicante/Alacant", ...  }
```

2) El script del servidor utilizado para cargar los municipios se llama `cargaMunicipiosJSON.php` y la respuesta del servidor tiene el siguiente formato:

```
{ "0014": "Alegría-Dulantzi",  "0029": "Amurrio",  ...  }
```

de librosweb.es/ajax

Consumo de APIs

- Google Maps fue una de las primeras aplicaciones basadas en AJAX de uso masivo por parte de los usuarios
- Antes de utilizar la API de los mapas de Google, es necesario obtener una clave personal y única para cada sitio web donde se quiere utilizar. El uso de la API es gratuito para cualquier aplicación que pueda ser accedida libremente por los usuarios. La clave de la API se puede obtener desde: https://developers.google.com/maps/

```
<script src="https://maps.googleapis.com/maps/api/js?v=3.exp&signed_in=true"></script>
<script>
var directionsDisplay;
var directionsService = new google.maps.DirectionsService();
var map;

function initialize() {
  directionsDisplay = new google.maps.DirectionsRenderer();
  var madrid = new google.maps.LatLng(40.429436, -3.686765);
  var mapOptions = {
    zoom:9,
    center: madrid
  };
  map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
  directionsDisplay.setMap(map);
}
```

Consumo de APIs

- Ejercicio de Información metereológica **EJ5**
- Tenemos un servidor cuyo script es previsionMeteorologica.php que devuelve en JSON con la lista de puntos geográficos junto con su previsión meteorológica

```
[
{ latlon: [42.779275360242, -2.63671875], prediccion: "tormentas" },
{ latlon: [43.245202722034, -8.32763671875], prediccion: "nieve" },
{ latlon: [42.228517356209, -7.36083984375], prediccion: "lluvia" },
...
{ latlon: [41.54147766679, -3.75732421875], prediccion: "nublado" },
]
```


Autocompletar

- Algunas veces, se presenta al usuario un cuadro de texto en el que tiene que introducir un valor que pertenece a un grupo muy grande de datos. una dirección de correo electrónico que pertenezca a la libreta de direcciones, el nombre válido de un municipio, el nombre de un empleado de una empresa grande, un código, etc
- Utilizar una lista desplegable que muestre todos los valores puede ser es completamente inviable, ya que pueden existir miles de posibles valores. Por otra parte, un cuadro de texto simple resulta de poca utilidad para el usuario
- La solución consiste en combinar un cuadro de texto y una lista desplegable mediante AJAX. El usuario escribe en el cuadro de texto, la aplicación solicita al servidor aquellos términos que estén relacionados con lo escrito por el usuario y los presenta en la lista a modo de ayuda para el usuario