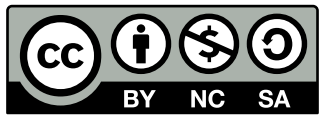
# AJAX / Escalabilidad de aplicaciones web

Jesús Arias Fisteus

**Aplicaciones Web (OpenCourseWare, 2023)**

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática



Parte I

# AJAX (Asynchronous JavaScript and XML)

# AJAX (Asynchronous JavaScript and XML)

- Nombre que se aplica al uso combinado de **JavaScript** y la API **XMLHttpRequest** para realizar peticiones HTTP en segundo plano, sin necesidad de cargar de nuevo la página completa.
- Este nombre se ha quedado desfasado, dado que se prefiere el uso de **JSON (JavaScript Object Notation)** en vez de XML.

# Programación de HTTP desde JavaScript

- En principio, es el navegador el que genera peticiones HTTP y procesa las respuestas.
- Inicialmente JavaScript podía hacer que el navegador generase peticiones HTTP estableciendo el atributo **`src`** en **`img`**, **`iframe`** y **`script`**, pero esto tenía problemas de portabilidad entre navegadores.
- La API XMLHttpRequest permite de forma más sencilla a programas JavaScript que se ejecutan en el lado del cliente realizar peticiones HTTP y procesar sus respuestas.

# Uso de XMLHttpRequest

- Proceso de tres etapas:
  1. Creación del objeto XMLHttpRequest.
  2. Especificación y envío del mensaje HTTP al servidor.
  3. Recepción (síncrona o asíncrona) de la respuesta del servidor.
- El cuerpo de los mensajes intercambiados suele representarse con JSON, texto plano, HTML o XML.

# Peticiones asíncronas

- Se envían los datos con el método **`send`**, que retorna inmediatamente, sin esperar a que llegue la respuesta.
- Se registra una función de *callback* que será invocada por el navegador cada vez que cambia el estado de la petición (propiedad **`readyState`**):
  - **`readyState == 0`**: sin inicializar.
  - **`readyState == 1`**: conexión establecida.
  - **`readyState == 2`**: petición recibida.
  - **`readyState == 3`**: respuesta en proceso.
  - **`readyState == 4`**: respuesta recibida.

# Ejemplo de petición asíncrona

```
1  var request = new XMLHttpRequest();
2
3  // establecimiento de una función de callback
4  request.onreadystatechange = function() {
5      if(request.readyState == 4) {
6          if(request.status == 200) {
7              alert("Received: " + request.responseText);
8          } else {
9              alert("Error: returned status code " + request.
                   status
10                   + " " + request.statusText);
11         }
12     }
13 };
14
15 // especificación de método, URL y petición asíncrona
16 request.open("GET", url, true);
17
18 // envío (sin cuerpo de la petición por ser GET)
19 request.send(null);
```

# Ajax con JQuery

- JQuery proporciona una API de alto nivel sobre XMLHttpRequest, con funciones más cómodas para el programador:
  - Una función de bajo nivel: `$.ajax()`
  - Un método de alto nivel: `load()`
  - Cuatro funciones de alto nivel: `$.getScript()`, `$.getJSON()`, `$.get()` y `$.post()`.

# Método load

```
1  // Carga el documento como contenido del elemento "#a":
2  $("#a").load("article.html");
3
4  // Permite seleccionar un fragmento del documento:
5  $("#new-projects").load("/resources/load.html #proj li");
6
7  // Se puede registrar un manejador que se ejecutará
8  // cuando se complete la operación:
9  $("#result").load("test.html", function() {
10   alert("Load was performed.");
11 });
12
13 // O cuando falle:
14 $("#success").load("/not-here.php",
15   function(response, status, xhr) {
16     if (status == "error") {
17       var msg = "Sorry but there was an error: ";
18       $("#error").html(msg + xhr.status
19                            + " " + xhr.statusText);
20     }
21 });
```

# Función $.getJSON()

```
1  // Carga un objeto JSON:
2  $.getJSON("test.json")
3    .done(function(json) {
4      // la variable json contiene el objeto de la respuesta
5      console.log("JSON␣Data:␣" + json.users[0].name);
6    })
7
8  // Se pueden pasar parámetros y gestionar errores:
9  $.getJSON("test.json", {name: "John", time: "2pm"})
10   .done(function(json) {
11     console.log("JSON␣Data:␣" + json.users[0].name);
12   })
13   .fail(function(jqxhr, textStatus, error) {
14     var err = textStatus + ",␣" + error;
15     console.log("Request␣Failed:␣" + err);
16 });
```

# Función $.get()

```
1 // Carga un objeto (lo interpreta automáticamente
2 // como JSON, XML, HTML o texto plano
3 // según corresponda):
4 $.get("test.php", {name: "John", time: "2pm"})
5   .done(function(data) {
6     $("body")
7       .append("Name: " + data.name)
8       .append("Time: " + data.time);
9   };
```

# Función $.post()

```
1  // Envía datos con POST
2  $.post("test.php", {name: "John", time: "2pm"});
3
4  // Envía un formulario con POST
5  $.post("test.php", $("#testform").serialize());
6
7  // También se puede procesar la respuesta HTTP
8  $.post("test.php", {name: "John", time: "2pm"})
9    .done(function(data) {
10     $("body")
11       .append( "Name: " + data.name )
12       .append( "Time: " + data.time );
13   };
```

# Referencias


- David Flanagan. “JavaScript: The Definitive Guide” (6th Ed.) O’Reilly (2011).
  - Capítulos 18 y 19.

Parte II

# Escalabilidad en aplicaciones web

# Escalabilidad en aplicaciones Web

El término **escalabilidad** se refiere a la capacidad de la infraestructura de servidor de una aplicación web de dar servicio a una carga creciente (tasa de peticiones, número de usuarios, etc.), o capacidad de ser ampliable para ello.

# Uso de memoria caché

- El uso de **memoria caché** reduce la carga del servidor web y del gestor de bases de datos:
  - No es necesario volver a construir páginas HTML que han sido construidas recientemente.
  - No es necesario volver a solicitar datos que se han consultado recientemente a la base de datos.

# Uso de memoria caché

- Uso de los mecanismos de control de caché de HTTP 1.1 y versiones posteriores.
- Uso de caché de páginas delante del servidor web (proxy inverso). P.e.: NGINX.
- Uso de sistemas distribuidos de caché en RAM. P.e. memcached:
  - Almacenamiento de fragmentos de HTML ya construidos.
  - Almacenamiento de objetos obtenidos de la base de datos.

# Excalar incrementando recursos

Cuando el uso de memoria caché no es suficiente, se pueden incrementar los recursos de cómputo.

# Dos formas de escalar incrementando recursos

- Escalabilidad vertical:
  - Se mejoran los equipos que ejercen de servidor web o gestor de base de datos (más y/o mejores CPUs, más RAM, más disco, etc.)
- Escalabilidad horizontal:
  - Se añaden más equipos (más servidores web o más gestores de bases de datos), sin necesariamente mejorar la capacidad de cada servidor.

# Escalado horizontal del servidor Web

- Se despliegan varios servidores web y se reparten las peticiones entre ellos:
  - Balanceo de carga por DNS: un mismo nombre de dominio con varias IPs.
  - Balanceo de carga con servidores de *front-end*.
- Aspectos a tener en cuenta:
  - Gestión de sesiones.

# Partición

- A veces es posible particionar tareas y/o datos en grupos:
  - Tareas: cada servidor web se encarga de unas tareas determinadas.
    - Cuidado con la gestión de sesiones.
  - Datos: cada gestor de bases de datos guarda un subconjunto de los datos (procurando que cada tarea se pueda llevar a cabo sobre un único gestor de bases de datos).

# Partición de datos

- Partición de datos:
  - Por temática de los datos. P.e. clientes en un gestor, ventas en otro, contabilidad en otro, etc.
  - Por subgrupos dentro de un mismo tema. P.e. subgrupos de clientes en distintos gestores, por identificador, ubicación, etc.

# Escalado horizontal de la base de datos

- Un servidor de bases de datos maestro, varios esclavos para lecturas:
  - Cuando el número de lecturas es bastante superior al de escrituras.

# Uso de bases de datos NoSQL

- Se relajan las restricciones ACID para distribuir la base de datos de forma más eficiente.
  - Bases de datos NoSQL.

# Bases de datos NoSQL

- Almacenamiento clave-valor:
  - Redis
  - DynamoDB
  - Berkeley DB

# Bases de datos NoSQL

- Basadas en columnas:
  - Cassandra
  - HBase (Hadoop)

# Bases de datos NoSQL

- Basadas en documentos:
  - MongoDB
  - CouchDB

# Bases de datos NoSQL

- Basadas en grafos:
  - Neo4J
  - OpenLink Virtuoso

# Referencias


- Martin L. Abbott; Michael T. Fisher. “Scalability Rules: Principles for Scaling Web Sites, 2nd Edition”. Addison-Wesley Professional (2016).
- Chander Dhall. “Scalability Patterns: Best Practices for Designing High Volume Websites”. Apress (2018).
  - Capítulos 1, 2 y 3.