SPA VS MPA

Historia del desarrollo Web

(1991-1995)

La primera página Web fue creada por Tim Berners-Lee en 1991, mediante una computadora **NEXT**, con el objetivo final de informar sobre la World Wide Web.

NEXT es la empresa que fundó Steve Jobs (fuera de Apple). Enfocada en la creación de potentes equipos empresariales y educacionales.

Las ideas de Next sirvieron para crear el sistema operativo conocido Mac OS.

Estáticas Las páginas web estaban hechas prácticamente solo con texto, con lenguaje de hipertexto HTML y con una muy limitada gama de colores. Es verdad que vistas con el tiempo pueden parecer algo rudimentarias, pero para su época eran algo extremadamente moderno.

Aparecieron en esos años nuevos elementos que dotarían a las páginas web de un look más vivo, y múltiples funciones, que harían de la navegación un placer antes desconocido.

iconos: pictogramas utilizados para representar archivos, carpetas, programas…

Imágenes de fondo: al poner imágenes de fondo, cada página web adquiría personalidad propia y la hacía más interesante visualmente.

Banners: pieza publicitaria que se insertaba en la página web pudiendo así promocionar marcas y productos.

Botones: uno de los elementos más usados en toda página web. Se usan para activar o desactivar funciones o programas de una web.

SPA VS MPA

(1996-2000)

Con las nuevas herramientas para diseñar páginas web, Los diseñadores empezaban a tener a su alcance la posibilidad de desarrollar páginas web mas interactivas.

Herramientas que hoy en día se usan de esta etapa se encuentran:

Flash: Flash revolucionó el diseño de páginas web gracias a la incorporación de contenidos multimedia y sus animaciones. Las dotaron de una vida antes desconocida.

Javascript: lenguaje de programación interpretado, orientado a objetos y dinámico, que permitía mejorar la navegación y la interfaz del usuario.

CSS: con la llegada del HTML3, llegaron las primeras hojas de estilos CSS , lo que mejoró la visibilidad de las páginas web.

Frames: permitían dividir la pantallas en pantallas más pequeñas, facilitando la carga de distintos contenidos.

Flash. Google y Apple no indexaban los contenidos de las páginas desarrolladas con tecnología Flash, lo que hizo que su uso fuera cada vez menor.

Siglo XXI

Se desarrollaron y evolucionaron las herramientas ya existentes, se potenció mucho el texto sobre la imagen y se crearon páginas web cada vez más orientadas al contenido (web 2.0).

CSS. Era un lenguaje que permitía trabajar por separado el contenido y el diseño, aportando una mayor libertad para poder diseñar la página web de un modo mucho más creativo.

También se desarrolló la idea de aportar espacios blancos en las páginas, para ofrecer descanso ocular a los usuarios.

A partir de 2013 se avanzó en el desarrollo de páginas web responsive, para poder ofrecer una correcta visualización y uso en todo tipo de dispositivos electrónicos.

Se desarrolló el estilo conocido como FLAT DESIGN. Atrás quedaron los relieves y el realismo, y se buscó una manera de diseñar estándar. Desde entonces todas las páginas se han desarrollado igual.

En la página flatvsrealism puedes ver una divertida comparativa sobre el cambio de tendencia.

SPA VS MPA

web hoy

En el 2016 aparece HTML5, PHP, ASP y otras como el Javascript se han desarrollado.

Entre los avances:

Multimedia: la aparición del HTML5 aportó la compatibilidad de elementos multimedia en cualquier navegador y dispositivos actuales.

Redes sociales: nos ofrecen plataformas virtuales en internet para que podamos relacionarnos con personas de cualquier lugar del mundo, y para compartir intereses, información o para simplemente perder el tiempo.

Contenidos interactivos: una atracción total para los usuarios. La variedad y la posibilidad de aprender con ellos los convierte en un aporte imprescindible.

Dispositivos móviles y tablets: llévate internet donde quieras y aprovechar así sus múltiples funciones y usos.

Aplicaciones: ligadas al punto anterior. Sin duda la gran aportación de los últimos 5 años y que van a seguir dando guerra.

La aparición de los distintos dispositivos electrónicos ha marcan la evolución del diseño web. Ahora las páginas web deben ser responsive y cargar lo más rápido posible.

futuro

El futuro del diseño web estará ligado al desarrollo de las herramientas existentes y su adaptación a todo tipo de tablets, móviles, televisiones, etc.

Los usuarios seguirán evolucionando y el diseño web también. Las aplicaciones y las funciones interactivas marcarán la evolución.

5 tendencias de diseño web:

Growth Drive desing: es el diseño web que promueve el crecimiento(plan, testeo, aplicación y difusión las mejoras) para esto se debe obtener información sobre los clientes, su navegación, sus gusto.

Animaciones y vídeo: hacen mas interactivo el sitio web.

Material design vs flat design:El flat design aporta elegancia y en cierta parte minimalismo a nuestra web y incluye la posibilidad de texturas, sombreados y pequeños detalles que enriquecen las imágenes.

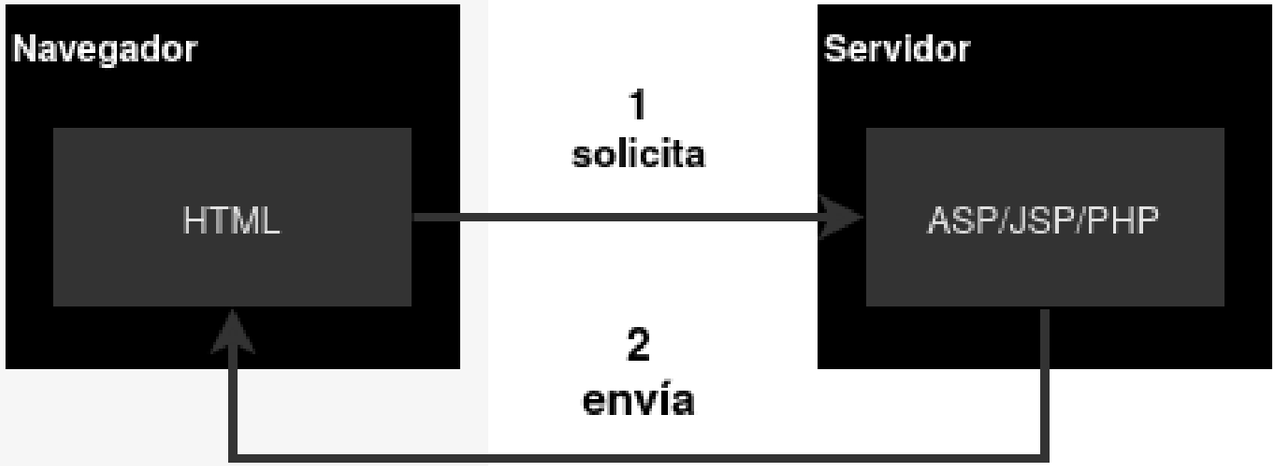
El formulario de registro bien visible: Incluir un formulario de registro

imágenes Hero: imágenes tan impactantes que ocupan toda la pantalla de la web

Arquitecturas Web y su evolución

El Modelo cero y el código spaguetti

Esta las soluciones iniciales **cliente/servidor** en las que tenemos una página **JSP/ASP/PHP** que se conecta a una base de datos y genera un nuevo contenido html. Como ventaja fundamental destaca su sencillez a nivel de arquitectura y como desventaja su poca flexibilidad y nula capacidad de reutilización.

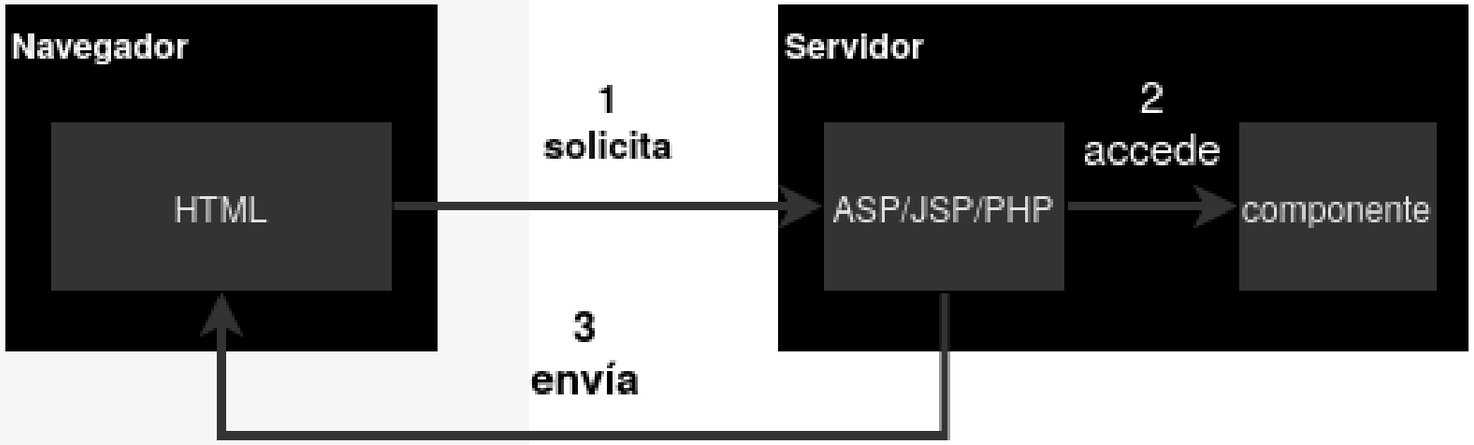


SPA VS MPA

Arquitecturas Web y su evolución

El Modelo 1

Promueve la modularización y el uso de componentes a través de la programación orientada a objeto. Fue desde mi punto de vista un gran salto. Hay enfoques tipo Rails que se apoyaron más en un patrón tipo Active Record y otros en Servicios y Repositorios como Spring. La clave fundamental es generar componentes en la capa de backend y aumentar la reutilización de esa parte.

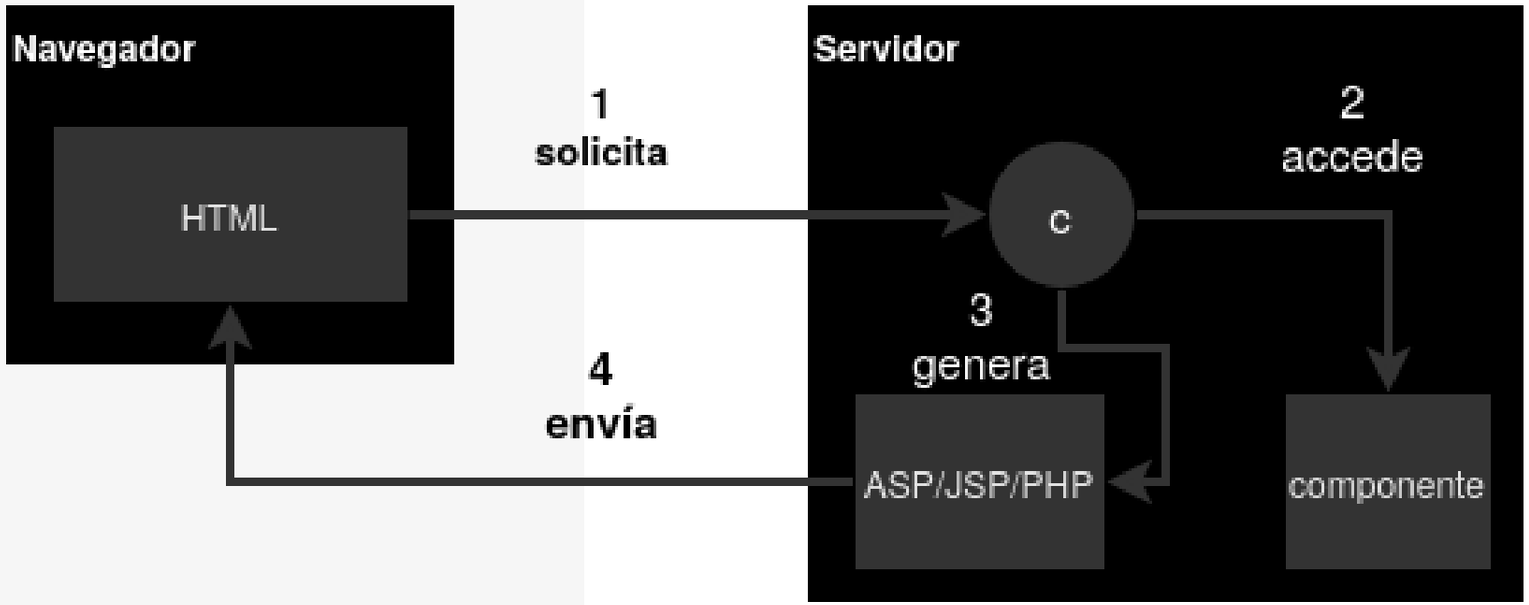


SPA VS MPA

Arquitecturas Web y su evolución

El modelo 2 (MVC)

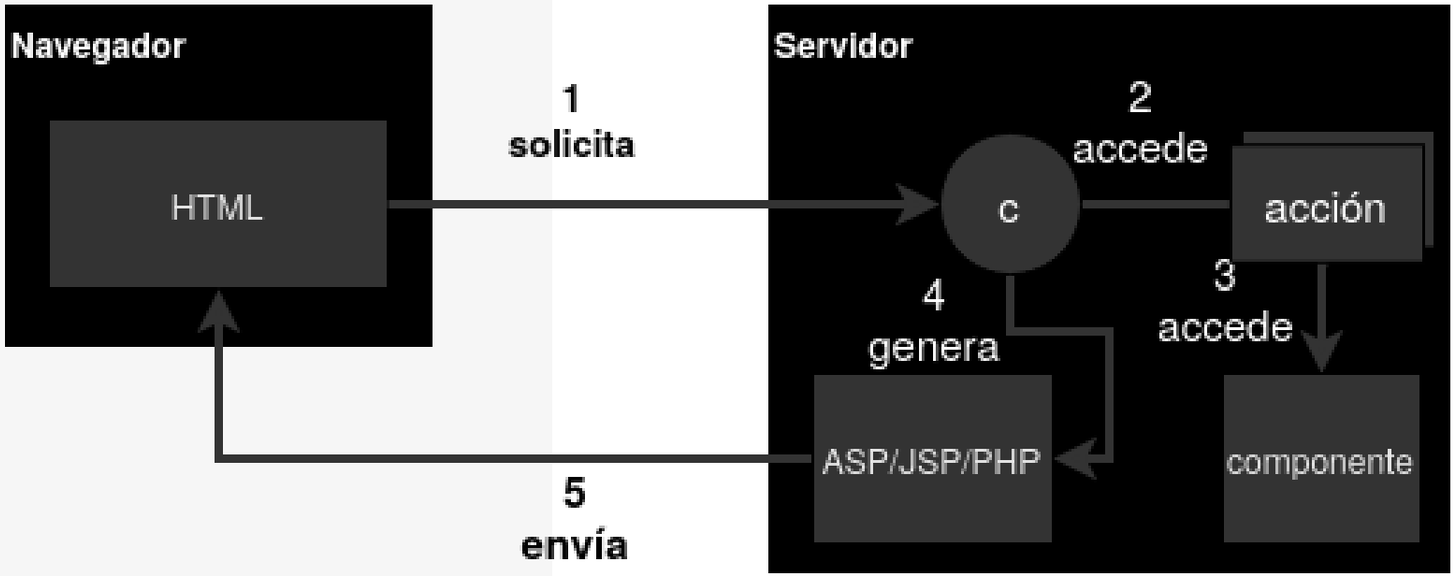
Se apuesta por la separación de responsabilidades entre Vista , Controlador y Modelo. Prácticamente todos los frameworks web han implementado este enfoque de una forma u otra. Ejemplos son: Struts en Java , ASP.NET en Microsoft o Laravel en PHP.



Arquitecturas Web y su evolución

El Modelo MVC 2 FrontController/Enrutador

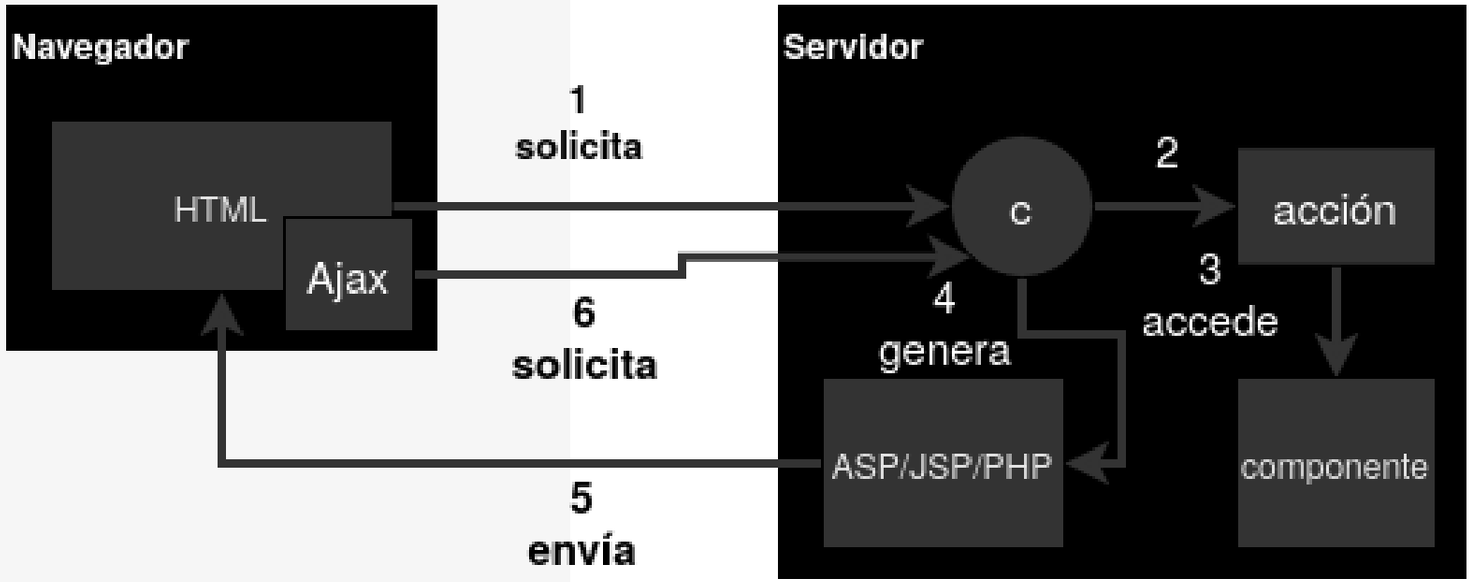
Una evolución importante del modelo MVC fue el modelo MVC 2 o de FrontController que apuesta por una arquitectura en la que únicamente hay un controlador principal y gestiona todo a través de acciones. Esta es un poco la idea de muchos de los frameworks modernos con el concepto de router en la capa de presentación. El enfoque más clásico puede ser Struts en el lado del servidor o Spring MVC.



Arquitecturas Web y su evolución

Arquitecturas Web y Ajax

Hasta ese momento todas las evoluciones se produjeron del lado del servidor. El lado cliente tenía pocas novedades. Es en esta situación cuando surge AJAX como tecnología para mejorar el rendimiento entre cliente y servidor. Esto supuso una verdadera revolución a la forma de programar.

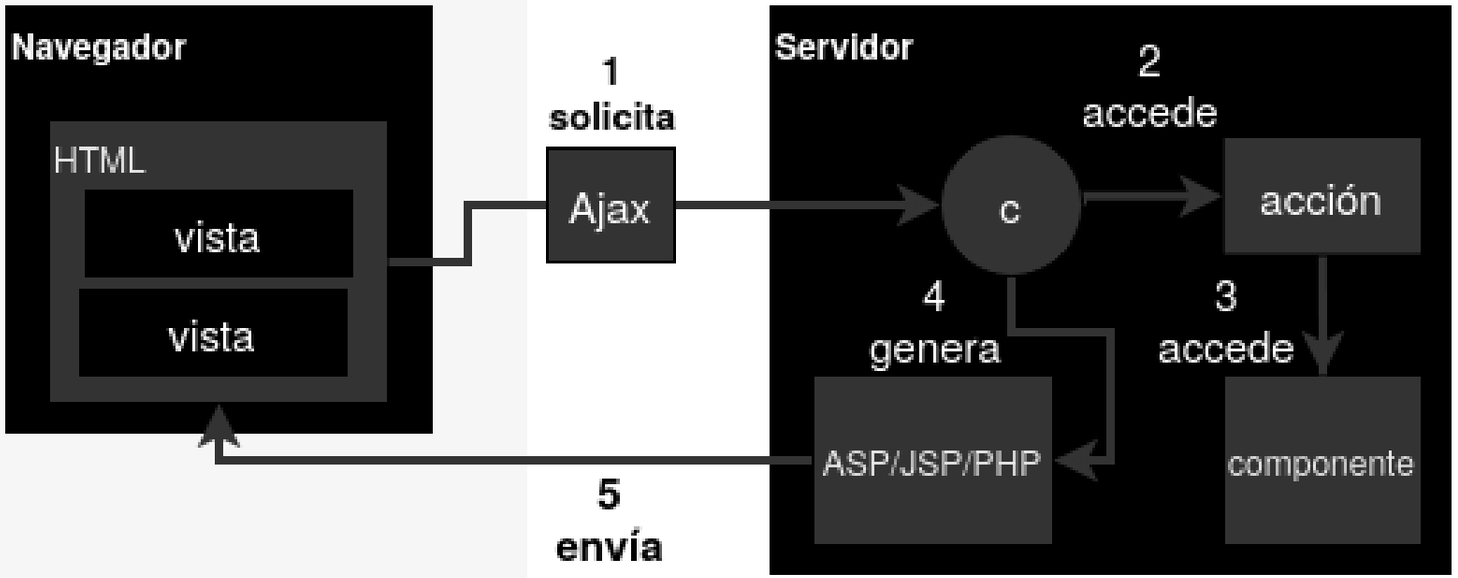


SPA VS MPA

Arquitecturas Web y su evolución

Arquitecturas Web y SPA

Con la llegada del mundo móvil y la necesidad de tener las aplicaciones web cada vez más desconectadas surgen las arquitecturas SPA (Simple Page Application) . Su propuesta principal es dar mayor responsabilidad al navegador y que el se encargue de cargar las vistas y datos utilizando AJAX.

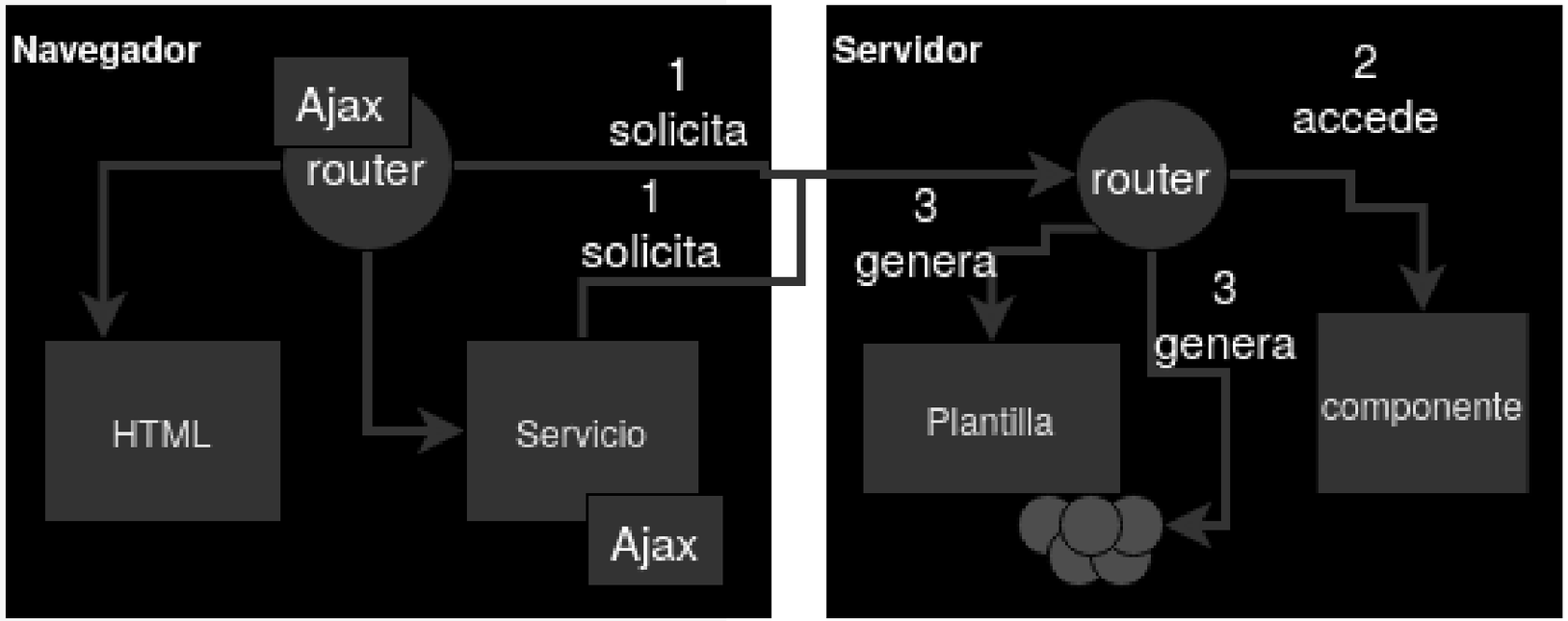


SPA VS MPA

Arquitecturas Web y su evolución

Arquitecturas SPA MVC

Poco a poco el lado cliente comienza a tener más peso en los desarrollos y se necesita organizar mejor el código de JavaScript . Aparecen los primeros frameworks MVC de cliente como Backbone.js que permiten dividir las responsabilidades de la misma forma que en el servidor.

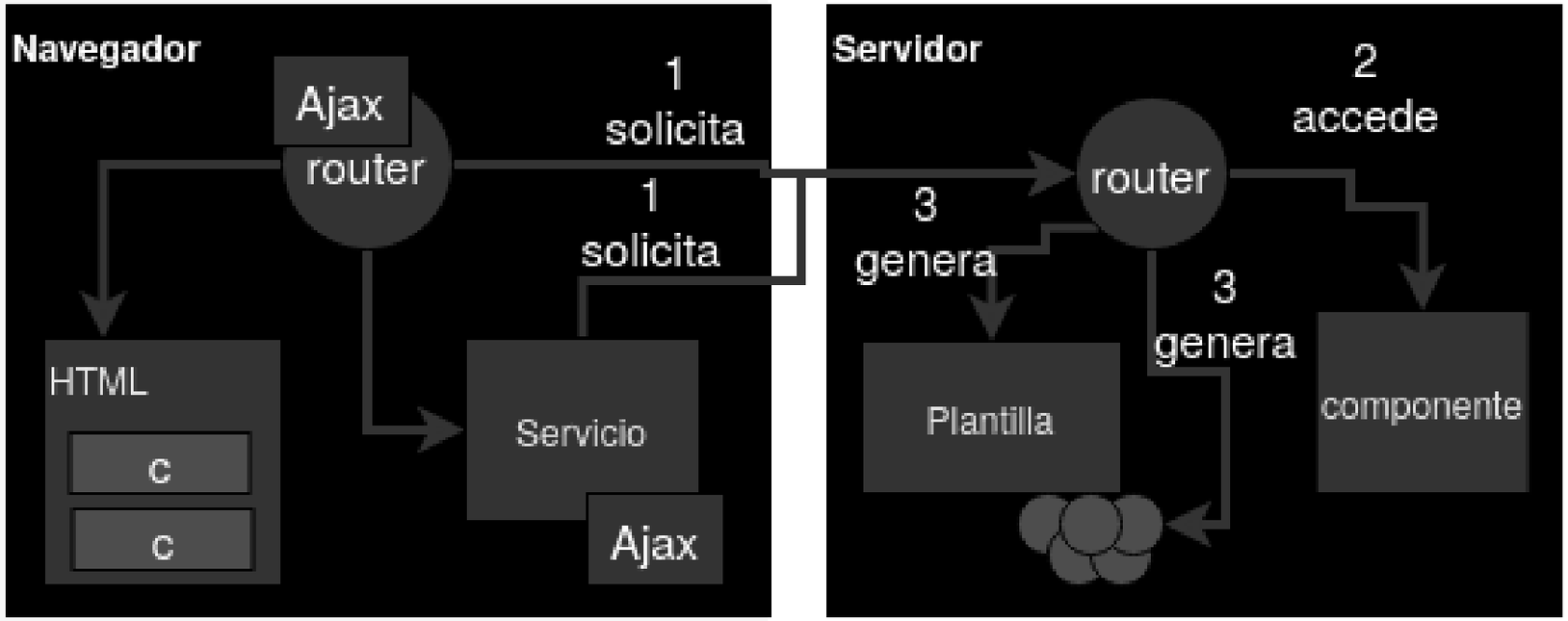


SPA VS MPA

Arquitecturas Web y su evolución

Arquitecturas SPA MVC y uso de componentes

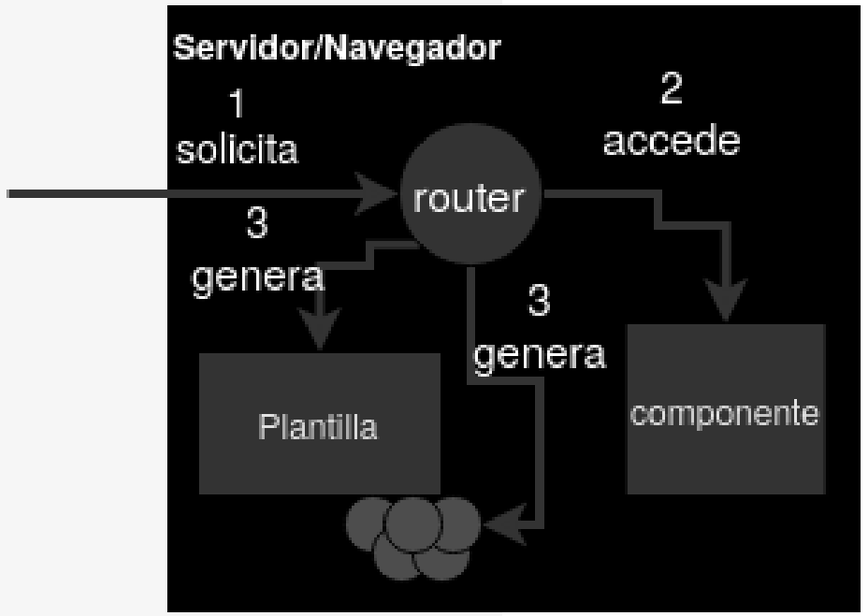
Estas arquitecturas empiezan a madurar rápidamente y aparecen tecnologías como Angular.js que promueve el uso del modelo MVC y la utilización de componentes en capa de presentación . Aparecen librerías complementarias como React que se centran en estos últimos.



Arquitecturas Web y su evolución

Arquitecturas Web Isomórficas

Ahora mismo estamos entrando en otra fase , comienza a llegar el JavaScript Isomórfico . Si nos fijamos en el último diagrama la parte cliente y la parte servidor son muy parecidas. ¿Qué sucedería en el caso de que ambas partes estuvieran implementadas en JavaScript? . Pues que probablemente mucho código se podría compartir y según se ejecutara la aplicación en cliente o en servidor el comportamiento variaría.



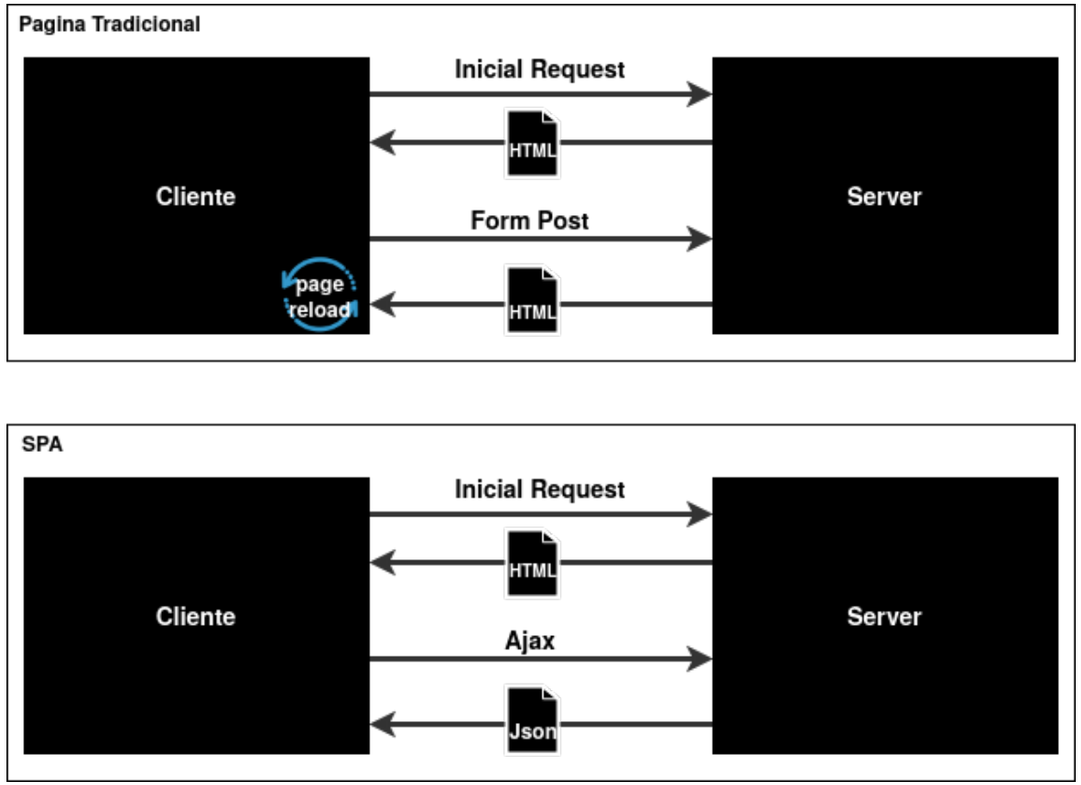
SPA VS MPA

Single Page App vs Multi Page App

El enfoque clásico es aquel en el que los clientes solicitan páginas y el servidor las construye en cada petición. Este enfoque es denominado como **“Page Redraw” o “Multipage Application”**. Aunque hoy en día, también estas webs con este enfoque tienden a apoyarse en peticiones **ajax** tratando de hacer el contenido más dinámico y agilizando la carga y experiencia de usuario creando un modelo híbrido.

Por otro lado nos encontramos con un nuevo enfoque alineado con las tecnologías **javascript** del cliente. Este enfoque que está marcando tendencia es aquel en el que el cliente solicita una única vez la web y el resto de peticiones no devuelven html, si no que se solicita el contenido al servidor vía **ajax** o **websocket** y se encarga de construir el html que se debe de visualizar con los datos recibidos. Este nuevo enfoque denomina **“Single Page Application”**.

Cada uno de los enfoques tiene sus ventajas e inconvenientes y no se puede determinar cual de los dos enfoques es el bueno. Cada proyecto, situación o necesidades van a marcar que tipo de arquitectura debemos plantear.



SPA VS MPA

Single Page App vs Multi Page App

“Page Redraw” o “Multipage Application”

Los navegadores web fueron diseñados para este tipo de aplicaciones clásicas, para las cuales existen cantidad de lenguajes y frameworks diseñados específicamente para construir este tipo de aplicaciones. Entre las ventajas que tiene este esquema es que son perfectas para SEO y existen recursos y herramientas que han trabajado con esta arquitectura permitiéndonos medir analíticas webs y haciendo un desarrollo mas controlado.

Sin embargo, aunque la gran ventaja es que es la arquitectura más empleada en los desarrollos webs hasta la fecha, han surgido nuevos enfoques para otras necesidades las cuales no requieren las necesidades que tenían las webs clásicas. La gran desventaja de este sistema es la recarga total de la web por cada contenido.

Hoy en día hay aplicaciones internas, que quizás no necesiten un sistema de rutas específico que permita una optimización de búsqueda en navegadores, si no lo que se busca es una integración con un servidor permitiendo una carga rápida, datos en tiempo real, notificaciones,etc… Básicamente que sea funcional con la necesidad del cliente.

SPA VS MPA

Single Page App vs Multi Page App

“Single Page Application”

Este nuevo enfoque implica aumentar la lógica en el lado del cliente, para la cual están comenzando a surgir nuevos frameworks que den soporte a este enfoque como pueden ser **AngularJs, Backbone, Reactjs**, entre otros.

Las ventajas de este enfoque, es que todos los datos están disponibles via **API**. Esto puede ser una gran ventaja por que exponer una api abre posibilidades de nuevas aplicaciones que puedan comunicarse con tu **backend**. Al ser totalmente independientes, la **API** puede crecer independientemente de la aplicación permitiendo un mantenimiento totalmente independiente.

Más ventajas que encontramos es que la aplicación tras cargar el estado y datos iniciales, permite mantenerse comunicada con pequeñas peticiones compactadas en formato **JSON** que les permite ser generalmente mas rápido y libera al servidor de bastante trabajo. Esto también implica una mayor experiencia de usuario evitando recargas pesadas e innecesarias y una interacción con la web en tiempo real.

Desventajas.

Ya que bastante lógica del negocio se implementa en el lado cliente, permitiendo que cualquiera pueda leerla en nuestro código javascript, por lo que hay que reforzar la seguridad y trabajar con minificadores js. También debemos de tener en cuenta que de esta manera hacemos trabajar mayormente al navegador. En términos de SEO tampoco es muy eficiente ya que el crawler de Google no soporta javascript volviendo mas complejo el SEO.

SPA VS MPA

Single Page App vs Multi Page App

	SPA	MPA
Sleek UX (exp Usuario)	✓	
Fácil SEO		✓
Seguridad		✓
Menor carga en el servidor	✓	
Funciona offline	✓	
Adaptabilidad Móvil	✓	
Aplicación Escalable		✓
Separación UI/Data	✓	
Velocidad	✓	
Despliegue Rápido		✓
Funciona sin JavaScript		✓

AJAX

Que es

El término AJAX se presentó por primera vez en el artículo "Ajax: A New Approach to Web Applications" publicado por Jesse James Garrett el 18 de Febrero de 2005. Hasta ese momento, no existía un término normalizado que hiciera referencia a un nuevo tipo de aplicación web que estaba apareciendo.

En realidad, el término AJAX es un acrónimo de **A**synchronous **Ja**vaScript + **X**ML.

Ajax se trata de varias tecnologías independientes que se unen. Las tecnologías que forman AJAX son:

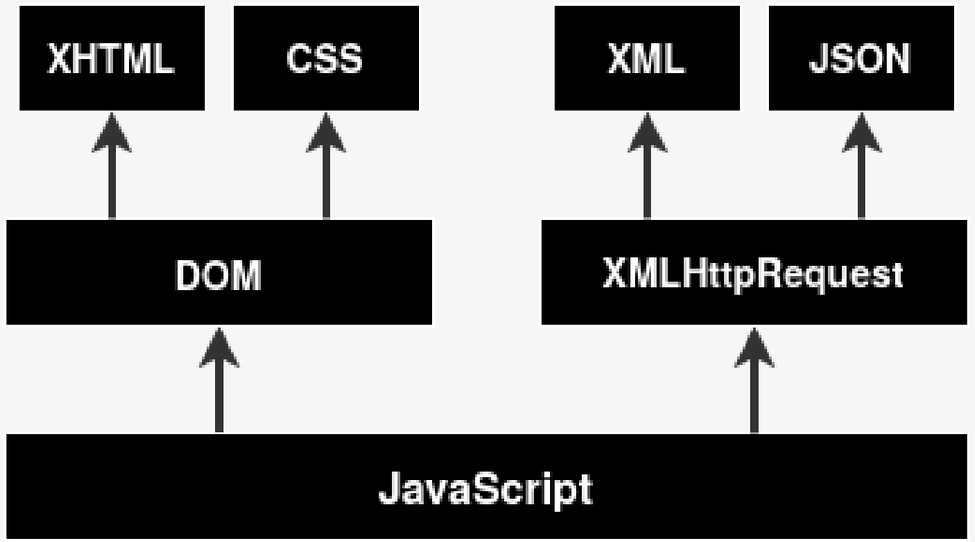
XHTML y CSS, para crear una presentación basada en estándares.

DOM, para la interacción y manipulación dinámica de la presentación.

XML, XSLT y JSON, para el intercambio y la manipulación de información.

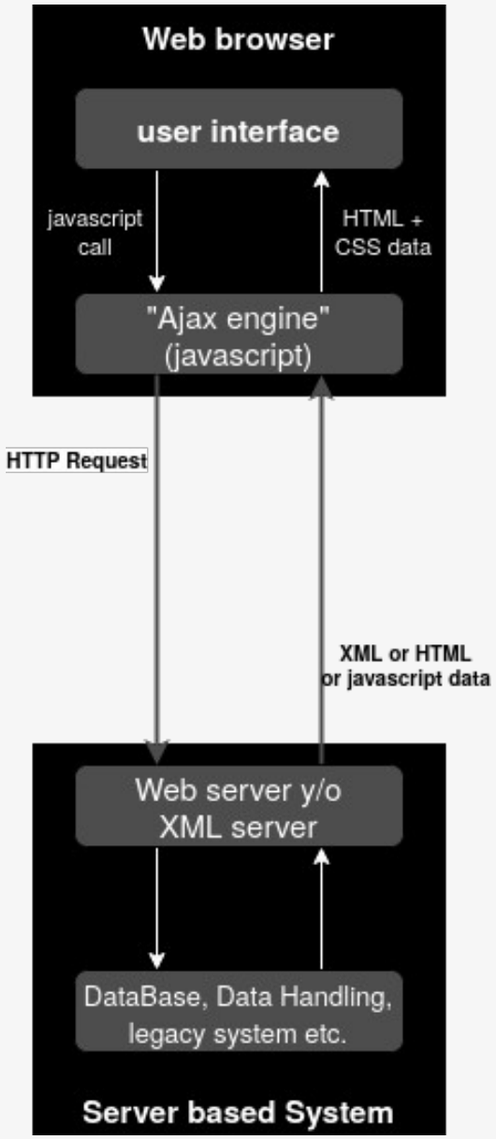
XMLHttpRequest, para el intercambio asíncrono de información.

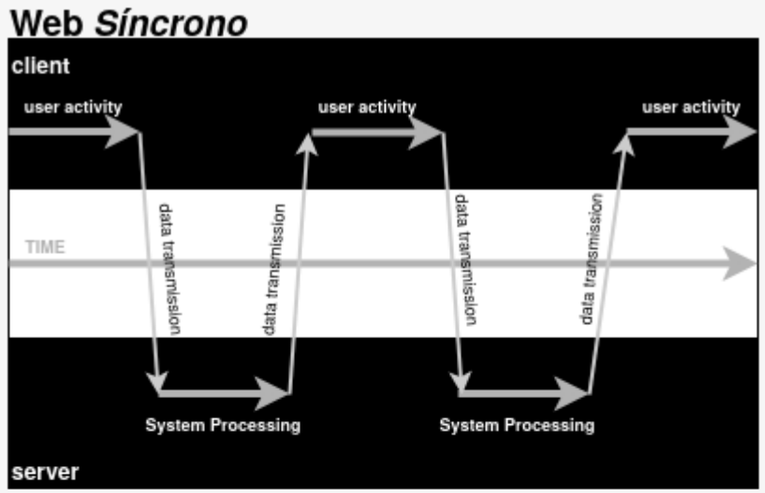
JavaScript, para unir todas las demás tecnologías.

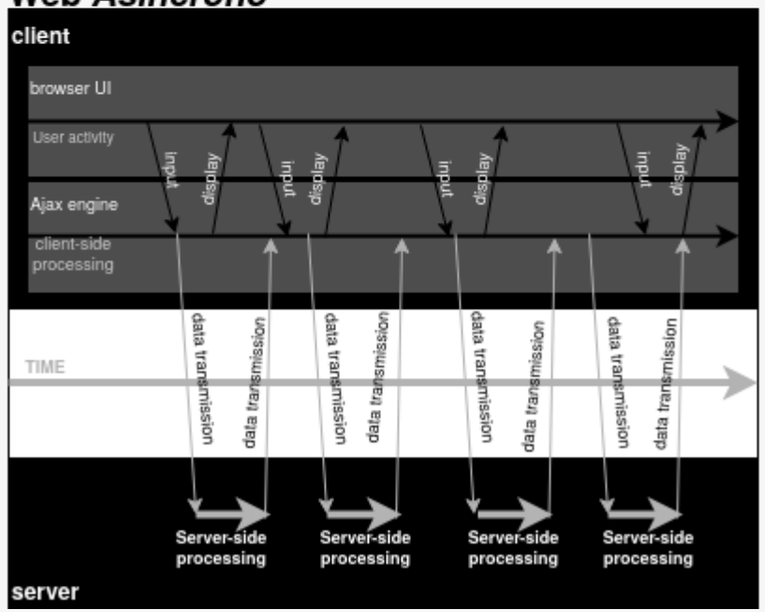


AJAX

Que es


Web browser
user interface
javascript call
HTML + CSS data
"Ajax engine" (javascript)
HTTP Request
XML or HTML or javascript data
Web server y/o XML server
DataBase, Data Handling, legacy system etc.
Server based System


Web Síncrono
client
user activity
user activity
user activity
TIME
data transmission
data transmission
data transmission
data transmission
System Processing
System Processing
server


Web Asíncrono
client
browser UI
User activity
Ajax engine
client-side processing
input
display
TIME
data transmission
Server-side processing
server

AJAX

Conceptos clave

GET vs. POST

El método **GET** debe ser utilizado para operaciones no-destructivas, operaciones en donde se esta obteniendo datos del servidor, pero no modificando.

El método **POST** debe ser utilizado para operaciones destructivas, operaciones en donde se está incorporando información al servidor. Por ejemplo, cuando un usuario guarda un artículo en un blog, esta acción debería utilizar POST

Tipos de datos

El tipo de dato es especificado por el nombre del método

text Para el transporte de cadenas de caracteres simples.

html Para el transporte de bloques de código HTML que serán ubicados en la página.

script Para añadir un nuevo script con código JavaScript a la página.

json Para transportar información en formato JSON, el cual puede incluir cadenas de caracteres, arrays y objetos

Asincronismo

Las llamadas Ajax son asíncronas, la respuesta del servidor no esta disponible de forma inmediata. código no funciona

```
var response;
$.get('foo.php', function(r) { response = r; });
console.log(response); // indefinido (undefined)
```

Es necesario especificar una función de devolución de llamada; dicha función se ejecutará cuando la petición se haya realizado de forma correcta ya que es en ese momento cuando la respuesta del servidor esta lista.

```
$.get('foo.php',function(response{ console.log(response); });
```

Políticas de mismo origen y JSONP

Ajax están limitadas a utilizar el mismo protocolo (http o https), el mismo puerto y el mismo dominio de origen. Esta limitación no se aplica a los scripts cargados a través del método Ajax de jQuery.

Ajax y Firebug

Firebug son herramientas para trabajar con peticiones Ajax.

Métodos Ajax de jQuery

método $.ajax en lugar de los otros, ya que ofrece más características y su configuración es muy comprensible.

$.ajax

Ofrece la posibilidad de especificar acciones en caso que la petición haya fallado o no. Además, al estar configurado a través de un objeto, es posible definir sus propiedades de forma separada, haciendo que sea más fácil la reutilización del código.

Opciones del método $.ajax

async Establece si la petición será asíncrona o no

cache Establece si la petición será guardada en la cache del navegador.

complete Establece una función de devolución de llamada que se ejecuta cuando la petición esta completa, aunque haya fallado o no

context Establece el alcance en que la/las funciones de devolución de llamada se ejecutara.

data Establece la información que se enviará al servidor.

dataType Establece el tipo de información que se espera recibir como respuesta del servidor

error Establece una función de devolución de llamada a ejecutar si resulta algún error en la petición.

jsonp Establece el nombre de la función de devolución de llamada a enviar cuando se realiza una petición JSONP.

success Establece una función a ejecutar si la petición a sido satisfactoria.

timeout Establece un tiempo en milisegundos para considerar a una petición como fallada.

type De forma predeterminada su valor es GET. Otros tipos de peticiones también pueden ser utilizadas (como PUT y DELETE).

url Establece la URL en donde se realiza la petición. La opción url es obligatoria para el método $.ajax;

Métodos Ajax de jQuery

Métodos convenientes

Los métodos que provee la biblioteca son:

$.get Realiza una petición GET a una URL provista.

$.post Realiza una petición POST a una URL provista.

$.getScript Añade un script a la página.

$.getJSON Realiza una petición GET a una URL provista y espera que un dato JSON sea devuelto.

Los métodos deben tener los siguientes argumentos, en orden:

url La URL en donde se realizará la petición. Su valor es obligatorio.

data La información que se enviará al servidor.

success callback Una función opcional que se ejecuta en caso que petición haya sido satisfactoria. .

data type El tipo de dato que se espera recibir desde el servidor. Su valor es opcional.

$.fn.load

El método $.fn.load es el único que se puede llamar desde una selección. Dicho método obtiene el código HTML de una URL y rellena a los elementos seleccionados con la información obtenida.

AJAX

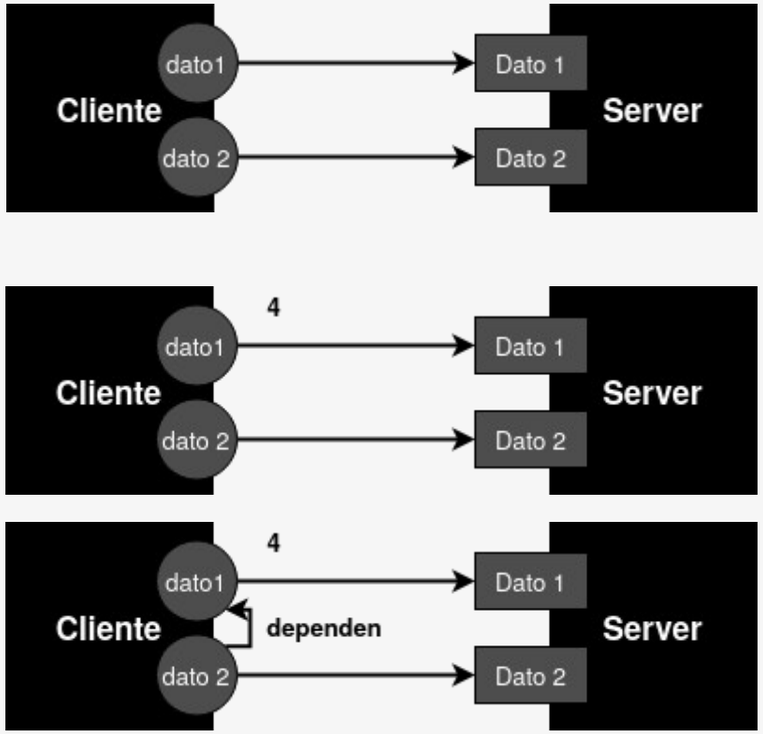
con JSONP

JSONP, ha permitido la creación de aplicaciones híbridas de contenidos. Muchos sitios importantes ofrecen JSONP como servicio de información, el cual se accede a través de una API

```
$.ajax({
    url : 'http://query.yahooapis.com/v1/public/yql',

     // se agrega como parámetro el nombre de la función de devolución,
    // según se especifica en el servicio de YQL
    jsonp : 'callback',

      // se le indica a jQuery que se espera información en formato JSONP
    dataType : 'jsonp',

    // se le indica al servicio de YQL cual es la información
    // que se desea y que se la quiere en formato JSON
    data : {
            q : 'select title,abstract,url from search.news where query="cat"',
        format : 'json'
    },

    // se ejecuta una función al ser satisfactoria la petición
    success : function(response) {
        console.log(response);
    }
});
```

jQuery Promise

Una Promise es un objeto que se ejecutará en un futuro. Todas las peticiones Ajax son Promesas. El concepto de promise pertenece al mundo de la programación asíncrona y no es exclusivo de jQuery.




```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <script type="text/javascript" src="js/jquery-1.11.3.min.js">
 </script>
 <script type="text/javascript">
 $(document).ready(function() {
  $("#boton").click(function() {
  var peticion1=$.get("http://localhost:8080/datos1",function(datos1) {
  console.log(datos1)
 });
  var peticion2=$.get("http://localhost:8080/datos2",function(datos2) {
  console.log(datos2)
 });
 });
 });
 </script>
 <title>Promesas</title>
</head>

<body>
 <input type="button" id="boton" value="pedirAjax"/>
</body>

</html>
```

```
var peticion1=$.get("http://localhost:8080/datos1",function(datos1) {

console.log(datos1);

var peticion2=$.get("http://localhost:8080/datos2",function(datos2) {

console.log(datos2);
});
});
});
```

jQuery Promise

$.when de jQuery que se encarga de controlar la ejecución de varias promesas y cuando todas hayan finalizado imprime la información requerida en el orden solicitado.

```
$("#boton").click(function() {

var promesa1=$.get("http://localhost:8080/datos1");
var promesa2=$.get("http://localhost:8080/datos2");

$.when (promesa1, promesa2).done(function (dato1,dato2) {

console.log(dato1[0]);
console.log(dato2[0]);
})

});
```

Métodos y propiedades del objeto XMLHttpRequest

Es un objeto JavaScript que fue diseñado por Microsoft. Actualmente es un estándar de la W3C. Proporciona una forma fácil de obtener información de una URL sin tener que recargar la página completa.

propiedades definidas

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.

Valores readyState

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)

Los métodos disponibles

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()

Modulo 2 (Ajax y Promesas) : Este modulo se centra en el manejo de promesas y como gestionar peticiones AJAX combinadas es decir en muchas ocasiones necesitamos realizar o anidar peticiones AJAX. Esto siempre genera situaciones complejas y veremos como solventarlas y abordar problemas como los de la pirámide de DOM.

Modulo 3 (Promesas Avanzadas): Este modulo se centra en abordar situaciones avanzadas con el manejo de Promesas . Muchas veces nos quedamos con los conceptos más elementales del manejo de estas. Sin embargo las promesas aportan mucha flexibilidad y es importante ver situaciones diferentes a la hora de utilizarlas

Modulo 4 (Ajax y REST):