Desarrollo de Front-END

Tema 9

AJAX

César Andrés Sánchez

Universidad Camilo José Cela
Escuela Politécnica Superior de Tecnología y Ciencia

Curso 2025 - 2026

AJAX

Ajax

Es un conjunto de técnicas para enviar información asíncrona entre servidor web y cliente

- Aunque hay antecedentes desde 1996, el Ajax actual lo crea google en 2004, para gmail y google maps
- Evita que el usuario necesite pulsar *enviar*. La aplicación web recibe información continuamente, resultando una experiencia de usuario similar a la de una aplicación de escritorio
- Originalmente significaba Asynchronous JavaScript and XML, pero cuando empieza a usarse sin JavaScript y sin XML, el acrónimo AJAX se abandona para usar la palabra Ajax

Funcionamiento de Ajax

1. El usuario solicita una URL desde su navegador
2. El servidor web devuelve la página, que contiene un script
3. El navegador presenta la página y ejecuta el script
4. El script hace peticiones HTTP asíncronas a una URI del servidor, sin que el usuario intervenga. El servidor suele ser RESTful
5. El script interpreta las respuestas HTTP y modifica el HTML, sin que el usuario intervenga

Same-origin policy

Same-origin policy es una norma que aparece en Netscape 2 (año 1995), que se ha convertido en un estándar. Consiste en que el código JavaScript solo puede acceder a datos que provengan del mismo origen desde el que se ha cargado el script

- Se entiende por *origen* el mismo protocolo, máquina y puerto

Ejemplo

- El usuario accede a una página web en *molamazo.com*,
- Esta página web puede tener código JavaScript que acceda a datos que estén en *molamazo.com*, pero solamente en este sitio
- No puede acceder a datos en *bancofuenla.es*
  - De lo contrario, una vez que el usuario se autentica en *bancofuenla* con una página de *bancofuenla*, un script malicioso en *molamazo* podría acceder a información sensible en *bancofuenla*

JSOP

JSONP (*JSON with padding, JSON con relleno*) es una técnica que permite que una página web obtenga datos desde un sitio web distinto al suyo, sin vulnerar la *same-origin policy*

- Es un protocolo del año 2005, soluciona el problema pero no es especialmente elegante. También tiene algunos problemas de seguridad potenciales
- Una alternativa más avanzada pero menos extendida es CORS, *Cross-origin resource sharing*

JSONP requiere de la colaboración del servidor. Es necesario que un servidor ofrezca datos no solo en JSON, también en JSONP

JSONP se basa en que la *same-origin policy* se aplica solo a datos, no a scripts

- Ejemplo: un script descargado desde *molamazo.com*, no puede descargar datos desde *bancofuenla.es*, pero sí puede descargar otro script
- Se entiende que esto no es especialmente peligroso. *Bancofuenla* podría enviar una contraseña a una página web, como un dato, esperando que la lea el usuario. Pero en un script nunca debería haber información sensible

Ejemplo de uso de JSOP

- Enviar 3 no está permitido, es un dato.
- Pero la cadena `"f(3)"` no es un dato. Es un script. Se puede enviar.
  - Si *bancofuenla* accede a enviar `"f(3)"` como dato JSON, sabe que esta información podría ser usada por un script de cualquier otro sitio, p.e. *molamazo*. Por tanto, nunca enviará de este modo información sensible
  - Un problema distinto, que JSONP no contempla, es que sea *bancofuenla* quien aproveche esto para enviar código dañino

En JSONP, el cliente solicita un dato a un sitio web (en nuestro ejemplo, *bancofuenla*), y también le indica el nombre de la función en la que se *envuelve* el dato.

- Esto se hace con un parámetro que suele denominarse `callback`
  Una petición podría ser
  `bancofuenla.es/divisas.html?par=USDEUR&fecha=hoy&callback=procesaDivisas`

- A la que el servidor podría responder
  `procesaDivisas(0.865)`
- `procesaDivisas()` será una función en el script cliente, que tratará el dato (0.865)

Ajax con jQuery

jQuery facilita mucho el uso de Ajax. El método principal es `ajax()`
Recibe:

- Un *plain object* que puede tener varios atributos. El principal es `url`, la dirección del servicio

Devuelve:

- Un objeto de tipo `jqXHR`, que tiene varios atributos. Los principales son:
  - Un método llamado `done` al que le pasamos la función que se invocará si la petición tiene éxito. Tendrá un parámetro, que en nuestro caso será un objeto JSON
  - Un método llamado `fail` al que le pasamos la función a invocar en caso de error

El siguiente ejemplo llama a `http://fixer.io`, un servicio que ofrece tipos de cambio de divisas

- Por motivos de seguridad, para cumplir con la *same-origin policy*, solo podemos cargar la página desde el disco duro local, no desde el web
- Si este servicio ofreciera respuestas en JSONP, sí podríamos usarlo normalmente. jQuery se ocupa del manejo de JSONP, resulta transparente para el programador
- Otra solución para los sitios sin soporte de JSONP (que son muchos), sería un proxy en el mismo sitio web que sirve el HTML

```
$(document).ready(function() {
  let urlServicio = 'http://data.fixer.io/latest';
  peticion = $.ajax({
    url: urlServicio,
    data: {
      access_key: "xxxxxx",
      symbols: "USD, GBP"
    }
  })
  peticion.done(manejaRespuesta);
  peticion.fail(manejaError);

  function manejaRespuesta(json) {
    $("#div01").text(JSON.stringify(json));
  };

  function manejaError(jqXHR) {
    $("#div01").text("Error: " + jqXHR.status);
  };
});
```

Cambio de divisas

- Cada divisa tiene un código ISO 4217, que es un identificador de 3 letras mayúsculas. Por ejemplo USD (United States Dollar) para el dólar, EUR para el euro, GBP para la libra esterlina, etc
- El precio de una divisa frente a otra se indica mediante la concatenación de sus dos códigos ISO 4217. La primera se denomina *divisa base* y la segunda, *divisa cotizada*
- Ejemplo EURUSD = 1.13
  Aquí la divisa base es el euro, y la cotizada, el dólar.
  Este valor (1.13) indica cuántas unidades de la divisa cotizada hacen falta para comprar una unidad de la divisa base
- Dicho de otro modo, la primera moneda es la que queremos y la segunda, la que tenemos, la que usamos para pagar.

setInterval

Para que la consulta Ajax se repita periódicamente, podemos usar la función `setInterval()`
Recibe

- Como primer argumento, una función
- Como segundo argumento, un intervalo de tiempo en milisegundos

Como resultado, evalúa la función periódicamente, con el intervalo especificado. El siguente ejemplo se ejecutaría cada 60 segundos

```
setInterval(function() {
                miTexto=actualizaTexto();
                $("#p01").text(miTexto);
            },
            60000
);
```