Mecanismes de programació asíncrona client-servidor

Xavier Garcia Rodríguez

Desenvolupament web en l'entorn client

Índex

Introducció

Originalment, per actualitzar els continguts d'una pàgina s'havia de recarregar completament: no era possible afegir nous continguts dinàmicament. Això provocava una gran despesa de recursos tant a la banda del client com a la del servidor. La solució a aquest problema és la utilització d'AJAX, un conjunt de tecnologies que permeten fer peticions asíncrones al servidor, processar-ne les respostes i actualitzar la pàgina.

Aquesta tecnologia permet crear aplicacions i pàgines web que només cal carregar una vegada (per exemple Facebook) i afegeixen nous elements a mesura que l'usuari selecciona diferents opcions o es desplaça pels continguts.

En aquesta unitat aprendreu a implementar les vostres pròpies solucions AJAX per connectar amb serveis web o llegir fitxers tant en dominis propis com de tercers, a fer servir la biblioteca jQuery per simplificar el vostre codi i a connectar amb fonts de dades obertes.

Primerament, a l'apartat **"Comunicació asíncrona amb JavaScript"**, es discutiran les dificultats que presenta provar el codi AJAX. Seguidament, es descriuran les característiques i funcionalitats de l'objecte `XMLHttpRequest` abans de passar a la descripció de les dues arquitectures de serveis web més utilitzades: SOAP i REST. A continuació, es tractaran mètodes alternatius per fer peticions sobre altres dominis, s'explicarà com accedir a dades obertes i es descriurà una altra tecnologia que permet establir connexions a través de la xarxa. A més a més, es mostraran i compararan les opcions de depuració de peticions de Google Chrome i Mozilla Firefox.

A continuació, a l'apartat **"Programació de la comunicació asíncrona amb jQuery"**, es tractarà la creació i enviament de peticions utilitzant la biblioteca jQuery, que exposa una interfície més clara, permet utilitzar-la per l'enviament de peticions AJAX i JSONP i simplifica la utilització de paràmetres. Se'n descriuran les opcions, com connectar a serveis web tant propis com de tercers i alguns mètodes d'ajuda per codificar els paràmetres que cal enviar.

Per assimilar correctament els coneixements que comprenen aquesta unitat és imprescindible fer els exercicis i activitats, així com provar els exemples allotjats en CodePen i consultar la documentació de l'objecte `XMLHttpRequest` i la biblioteca jQuery en cas de dubte.

Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Desenvolupa aplicacions web dinàmiques, reconeixent i aplicant mecanismes de comunicació asíncrona entre client i servidor.

- Avalua els avantatges i els inconvenients d'utilitzar mecanismes de comunicació asíncrona entre client i servidor web.
- Analitza els mecanismes disponibles per a l'establiment de la comunicació asíncrona.
- Utilitza els objectes relacionats.
- Identifica les seves propietats i els seus mètodes.
- Utilitza comunicació asíncrona en l'actualització dinàmica del document web.
- Utilitza diferents formats en l'enviament i en la recepció d'informació.
- Programa aplicacions web asíncrones de manera que funcionin en diferents navegadors.
- Classifica i analitza llibreries que facilitin la incorporació de les tecnologies d'actualització dinàmica a la programació de pàgines web.
- Crea i depura programes que utilitzin aquestes llibreries.

1. Comunicació asíncrona amb JavaScript

Als inicis d'internet, cada vegada que un usuari realitzava una acció, la pàgina on es trobava s'havia de recarregar completament perquè reflectís aquests canvis (per exemple, quan s'afegia un ítem en un carretó electrònic).

Això suposava una mala experiència a l'usuari, perquè s'havia d'esperar que la pàgina es descarregués un altre cop sense poder interactuar-hi, per mínim que fos el canvi.

Descàrrega de fitxers i data d'expiració

Habitualment, en obrir una pàgina web, els navegadors emmagatzemen fitxers descarregats per no haver de tornar-los a recarregar en visites posteriors (imatges, codi CSS, codi JavaScript, etc.). La data d'expiració –si n'hi ha– depèn de la configuració del servidor, tot i que aquests fitxers es poden eliminar manualment del navegador.

Per altra banda, la càrrega del servidor també s'incrementava perquè havia de tornar a enviar tots els continguts per a cada petició que es feia, i en molts casos s'havien de fer consultes a la base de dades per tornar a generar els mateixos continguts que ja havia descarregat l'usuari inicialment.

1.1 Introducció a AJAX

Per resoldre el problema de la càrrega dinàmica de dades es fa servir AJAX (Asynchronous JavaScript and XML), un conjunt de tecnologies que permeten actualitzar els continguts d'una pàgina o aplicació web sense necessitat de recarregar-la. Les tecnologies que la componen són les següents:

- HTML i CSS per a la representació.
- El model d'objectes del document (DOM) per mostrar i manipular les dades dinàmicament.
- L'objecte `XMLHttpRequest`, que permet la comunicació asíncrona.
- Els formats JSON o XML per a l'intercanvi de dades.
- JavaScript per lligar totes aquestes tecnologies.

Com es pot apreciar, totes aquestes tecnologies estan integrades amb JavaScript i, consegüentment, no és necessari fer servir cap llibreria per treballar-hi (tot i que és habitual fer servir jQuery perquè d'aquesta manera no cal implementar la gestió de les peticions).

"Parsejar" i "Analitzar sintàcticament"

La utilització del terme *parsejar* és incorrecte però és possible trobar materials que ho fan servir en lloc d'*analitzar sintàcticament*. Cal tenir en compte que en tots dos casos es fa referencia a la mateixa operació.

Cal destacar que, tot i que originalment es va utilitzar XML per a l'intercanvi de dades, no és un requisit: es poden fer servir diferents formats perquè, com a resposta, admeten tant continguts en format XML com continguts en text pla que es poden analitzar sintàcticament (aquesta acció es coneix com a *parse* en anglès) per convertir-los a altres formats, per exemple, a JSON o fins i tot HTML.

L'objecte `XMLHttpRequest` permet realitzar peticions en segon pla (són asíncrones), de manera que l'aplicació no queda bloquejada quan es fa la petició, i una vegada es rep la resposta (amb dades o amb un missatge d'error) és processada per l'aplicació.

Fer servir aquesta tecnologia també permet crear aplicacions `single-page`, que consisteixen en una única pàgina a la qual es mostren uns continguts o uns altres segons les accions de l'usuari, sense haver de sortir-ne, o les pàgines amb desplaçament vertical (*scroll*) infinit, en què una vegada s'arriba al final de la pàgina es carreguen més continguts (com per exemple el llistat de notícies de Facebook o la cerca d'imatges de Google).

Un ús molt freqüent és la càrrega dinàmica de les dades de les llistes desplegables; per exemple, quan se selecciona una província d'un desplegable és habitual que es realitzi una petició al servidor que retorna el llistat de localitats que són afegides en una altra llista. D'aquesta manera, s'evita haver de descarregar totes les localitats cada vegada que un usuari visita la pàgina.

Exemple de carrega de fitxers mitjançant AJAX

El joc IOC Invaders (www.goo.gl/fMVSbs) fa servir AJAX per carregar la informació dels nivells i els enemics. Podeu trobar el codi font a www.goo.gl/LNiK94.

Un altre exemple d'ús seria la càrrega de nivells a un joc HTML, en el qual la informació s'obté d'un fitxer amb format JSON o XML, de manera que el client descarrega el motor del joc i aquest realitza les peticions necessàries per carregar els nivells, la informació sobre enemics o el rànquing. Això facilita molt la tasca de desenvolupar aquest tipus de continguts, ja que es poden crear editors independents (fins i tot en altres llenguatges) i fa possible ampliar el joc sense haver de tocar-ne la implementació.

Tot i que la connexió entre l'aplicació i el servidor s'inicia i acaba amb la petició, és possible crear sistemes de notificacions o de comunicació (tipus xat) realitzant peticions de forma continuada, fent servir els temporitzadors `setTimeout` i `setInterval`. Cal tenir en compte que aquest sistema no és gens eficient i en aquests casos és recomanable fer servir altres tecnologies com els Web Sockets, que permeten establir una connexió directa amb el servidor.

A banda de les peticions que podeu fer vosaltres directament, algunes llibreries i paquets de desenvolupament (com l'API de Google Maps) poden realitzar les seves pròpies peticions: per aquesta raó, en carregar una pàgina web no és gens estrany que hi hagi peticions fetes per tercers.

1.2 Dificultats a l'hora de provar el codi AJAX

Abans de començar a desenvolupar aplicacions que utilitzen AJAX s'ha de tenir en compte que hi ha algunes dificultats a l'hora de comprovar-ne el funcionament que cal conèixer.

Política del mateix origen

Es pot trobar més informació a l'enllaç següent: www.goo.gl/KgoIFn.

Com a mesura de seguretat, els navegadors apliquen la *política del mateix origen* (*same-origin policy*), que només permet realitzar peticions asíncrones dintre del mateix origen, és a dir, tant l'aplicació que realitza la petició com l'URL al qual s'envia han d'originar-se al mateix domini. No està permès fer aquestes peticions, ni tan sols entre subdominis, per tant, si una aplicació que es troba a `www.example.com` fa una petició a `api.example.com` el navegador la bloquejarà.

Primer de tot creeu un fitxer amb el següent codi HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script>
    // Escriviu aquí el vostre codi
    </script>
  </head>
  <body>
  </body>
</html>
```

I seguidament afegiu el següent codi dintre de les etiquetes `script`:

```
if (window.XMLHttpRequest) {
  httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) {
  httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
  console.log("Objecte creat a partir d'ActiveXObject.");
} else {
  console.error("Error: Aquest navegador no admet AJAX.");
}

httpRequest.onreadystatechange = function () {
  console.log("Ha canviat l'estat de la petició");
}

httpRequest.open('GET', 'http://www.example.com', true);
httpRequest.send(null);
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wzRjZy?editors=0012.

Aquest codi és completament correcte, però l'adreça de petició (establerta a httpRequest.open) indica que el domini al qual s'envia és `www.example.com`; així doncs, qualsevol petició enviada des d'altres dominis llançarà una excepció que es pot consultar a la consola de les eines de desenvolupador:

```
XMLHttpRequest cannot load http://www.example.com. No 'Access-Control-Allow-
    Origin' header is present on the requested resource. Origin 'http://s.
    codepen.io' is therefore not allowed access.
```

És possible evitar aquesta limitació fent servir altres mecanismes com CORS (*cross-origin resource sharing*; www.en.wikipedia.org/wiki/Cross-origin_resource_sharing), que afegeix noves capçaleres a la petició per restringir-ne l'accés als usuaris, o JSONP (JSON amb *padding*; www.ca.wikipedia.org/wiki/JSONP) per carregar la informació som si es tractés d'un script que seguidament és processat.

Cal destacar que alguns navegadors, com Google Chrome, inclouen el `localhost` dins de la política del mateix origen i, per tant, no funcionen correctament. Per aquest motiu **es recomana fer servir Mozilla Firefox** quan es desenvolupin aplicacions que utilitzen AJAX.

Així doncs, heu de tenir en compte que pràcticament cap dels exemples d'aquest material pot funcionar a CodePen, ja que tant el vostre codi JavaScript com el servidor al qual s'envia la petició han de trobar-se en el mateix domini i, segons la versió del navegador utilitzat, les vostres peticions locals també poden ser bloquejades. De totes maneres s'ha inclòs l'enllaç a CodePen dels exemples per facilitar "copiar i enganxar" el codi.

1.3 L'Objecte XMLHttpRequest

L'objecte `XMLHttpRequest` forma part de les especificacions del W3C i exposa una sèrie de mètodes i propietats que permeten gestionar les crides asíncrones i les respostes obtingudes.

XMLHttpRequest: propietats i mètodes

Podeu trobar una llista completa de propietats i mètodes de la interfície XMLHttpRequest a l'enllaç següent: www.goo.gl/1WpgA3.

Cal destacar que la major part de les propietats que formen part de la interfície `XMLHttpRequest` només són de lectura, ja que contenen informació que s'hi afegeix externament com a resultat de la petició. A continuació podeu trobar una llista de les més utilitzades:

- **onreadystatechange**: tot i que es tracta d'una propietat, la seva funció és assignar-n'hi una que serà cridada automàticament quan es produeixi un canvi a la propietat `readyState`.
- **readyState**: retorna l'estat de la petició.
- **responseText**: retorna la resposta de la petició com a text.
- **responseXML**: retorna la resposta de la petició com a XML.

Fixeu-vos que la resposta es pot obtenir tant en text pla com en XML. Així doncs, si el format de la resposta és diferent a XML, s'haurà d'obtenir de la propietat `responseText` i seguidament analitzar-la sintàcticament per convertir-la en el format correcte (per exemple a JSON).

Quant als mètodes disponibles per realitzar les peticions bàsiques hi ha els següents:

- **open**: inicialitza la petició.
- **overrideMimeType**: permet sobreescriure el tipus multimèdia de la resposta rebuda (`text/html`, `text/plain`, `application/xml`, etc.).
- **send**: envia la petició amb les dades passades com a paràmetre que poden ser, entre d'altres, una cadena de text, un diccionari de dades o tot el document.

Respecte a la detecció d'*events*, els navegadors moderns admeten l'ús del mètode `addEventListener`, així com l'ús de les dreceres següents:

- **onload**: drecera per afegir un detector de l'*event* `load`, que es dispara en rebre la resposta.
- **onerror**: drecera per afegir un detector de l'*event* `error`, que es dispara si es produeix algun error.
- **onprogress**: drecera per afegir un detector de l'*event* `progress`, que es dispara quan s'actualitza el progrés de la resposta i permet controlar la proporció rebuda respecte al total.

1.3.1 Creació i enviament de peticions

El primer pas per poder realitzar una petició AJAX és determinar si el navegador admet l'objecte XMLHttpRequest o no. En cas que no l'admeti es comprova si existeix l'objecte `ActiveXObject` (el seu antecessor, creat per Microsoft) i, si tampoc es troba, es mostra un missatge d'error:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script>
      if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+
        httpRequest = new XMLHttpRequest();
        console.log("Creat l'objecte a partir de XMLHttpRequest.");
      } else if (window.ActiveXObject) { // IE 6 i anteriors
        httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        console.log("Creat l'objecte a partir d'ActiveXObject.");
      } else {
        console.error("Error: Aquest navegador no admet AJAX.");
      }
    </script>
  </head>

  <body>
  </body>
</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/VKqxkv?editors=0012.

Una vegada s'ha creat l'objecte, a partir de *XMLHttpRequest* o d'*ActiveXObject*, es pot assignar una funció al mètode `onreadystatechange`, que serà invocada automàticament quan canvia l'estat de la petició; és on es gestiona què s'ha de fer amb la resposta o si es produeix un error.

Una vegada assignada la propietat `onreadystatechange` es poden fer servir els mètodes `open` i `send` per crear la petició i enviar-la:

- **open** (mètode, URL, asíncrona): proporciona la informació per crear la petició. El primer paràmetre determina el mètode que es farà servir per a la petició (`GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE`), el segon és l'URL en el qual es realitzarà la petició, i el tercer (opcional) indica si es vol fer la invocació asíncronament (true) o síncronament (false), cosa que bloquejaria l'execució fins a rebre'n la resposta. Aquest últim paràmetre és opcional, i si s'omet, la petició es fa de manera asíncrona.
- **send** (params): envia la petició prèviament preparada amb els paràmetres especificats. Aquests paràmetres han d'estar en un format que pugui interpretar el servidor de destí.

Per realitzar proves bàsiques amb AJAX no cal tenir un servidor funcionant, és possible crear els fitxers de prova manualment en una carpeta que sigui accessible des de la pàgina (com les imatges i el codi CSS i JS).

Afegiu dins de la mateixa carpeta del vostre codi un fitxer anomenat `provincies.xml` amb el contingut següent:

```
<provincies>
  <provincia>Barcelona</provincia>
  <provincia>Girona</provincia>
  <provincia>Lleida</provincia>
  <provincia>Tarragona</provincia>
</provincies>
```

A més a més, cal inserir el següent codi a l'exemple anterior:

```
httpRequest.onreadystatechange = processarCanviEstat;

httpRequest.open('GET', 'provincies.xml', true);
httpRequest.send(null);

function processarCanviEstat() {
  console.log("Ha canviat l'estat de la petició");
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wzRXmL?editors=0012.

Com es pot apreciar, una vegada es rep la resposta s'invoca la funció `processarCanviEstat`. És aquí on s'ha d'implementar la lògica per realitzar una acció o altra en funció de si s'ha finalitzat amb èxit o s'hi ha produït un error.

Alguns navegadors mostren un error a la consola si no s'especifica el joc de caràcters (`charset`) que ha d'utilitzar la pàgina, i si s'utilitzen caràcters no anglesos, no es mostren correctament, per tant, és recomanable especificar-ho.

En cas que el fitxer no tingui un format correcte també es mostrarà error, ja que per defecte s'espera que la resposta siguin dades en format XML. Així doncs, cal assegurar-se, per al bon funcionament dels exemples, que el fitxer XML és correcte.

El primer que s'ha de fer quan l'estat canvia és comprovar si ja s'ha completat o no. L'objecte XMLHttpRequest es pot trobar en 5 estats diferents i el seu valor s'obté a partir de la propietat `readyState`. Els noms dels valors possibles (i el seu valor) són els següents:

- **UNSENT** (0). Encara no s'ha obert la petició amb `open`.
- **OPENED** (1). S'ha obert la petició però encara no s'ha enviat.
- **HEADERS_RECEIVED** (2). S'ha enviat la petició.
- **LOADING** (3). S'està descarregant la resposta. La propietat `responseText` conté el contingut parcial.
- **DONE** (4). S'ha completat l'operació.

Cal destacar que a Internet Explorer els noms són diferents i, per consegüent, a l'hora de desenvolupar qualsevol aplicació, es recomana fer servir els valors enters i assignar-los com a pseudoconstants pròpies en lloc dels noms.

Substituïu la funció `processarCanviEstat` per la següent:

```
function processarCanviEstat() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    console.log("S'ha rebut la resposta. Estat actual:", httpRequest.readyState
        );
  } else {
    console.log("Encara no s'ha rebut la resposta... Estat actual:",
        httpRequest.readyState);
  }
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/mAajqy?editors=1012.

Com es pot apreciar, l'estat va canviant des de l'1 (`OPENED`) fins al 4 (`DONE`). Una vegada arribat a aquest punt es pot consultar la propietat `status` per comprovar si s'ha tingut èxit amb la petició o no. El valor d'aquesta propietat serà 200 quan no s'ha produït cap error o altres valors específics com 404 (pàgina no trobada) o 500 (error intern del servidor).

Substituïu la funció `processarCanviEstat` per la següent, a la qual s'ha afegit el codi per discriminar entre peticions amb èxit i peticions errònies:

```
function processarCanviEstat() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    console.log("S'ha rebut la resposta. Estat actual:", httpRequest.readyState
        );

    if (httpRequest.status === 200) {
      console.log("S'ha rebut la resposta correctament. Estat de resposta:",
          httpRequest.status);
```

```
    } else {
      console.log("S'ha produït un error en obtenir la resposta. Estat de
          resposta:", httpRequest.status)
    }

  } else {
    console.log("Encara no s'ha rebut la resposta... Estat actual:",
        httpRequest.readyState);
  }
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/QKzBZa?editors=1012.

Com es pot apreciar, primer s'ha de comprovar que el valor de `readyState` sigui 4 (`DONE`) i que l' `status` valgui 200 abans de processar la resposta.

Per acabar, només cal processar les dades i mostrar-les per pantalla, afegint la crida a la funció `processaResposta`, en cas d'èxit, i afegint-hi la nova funció:

```
function processarCanviEstat() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    console.log("S'ha rebut la resposta. Estat actual:", httpRequest.readyState
        );

    if (httpRequest.status === 200) {
      console.log("S'ha rebut la resposta correctament. Estat de resposta:",
          httpRequest.status);

      processarResposta(httpRequest.responseText);
    } else {
      console.log("S'ha produït un error en obtenir la resposta. Estat de
          resposta:", httpRequest.status)
    }

  } else {
    console.log("Encara no s'ha rebut la resposta... Estat actual:",
        httpRequest.readyState);
  }
}

function processarResposta(resposta) {
  var elementPre = document.createElement('pre');
  elementPre.innerHTML = resposta;
  document.body.appendChild(elementPre);
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/yaGxaR?editors=1012.

Com es pot apreciar, la funció `processarResposta` simplement crea un element i l'afegeix al cos del document, assignant la resposta com a contingut sense realitzar cap tractament, només per comprovar que funciona correctament.

Alternativament, en cas de treballar només amb navegadors moderns, es pot afegir la detecció de l'*event* `load` en lloc de controlar els canvis d'estat. Aquest mètode es va afegir a la segona versió de l'especificació i es dispara automàticament quan es completa la descàrrega. De la mateixa manera, es poden escoltar els *events* `error` i `progress` per determinar si s'ha produït un error o per actualitzar el progrés de la transmissió de dades, respectivament.

Cal recordar que com que es tracta d'*events* es pot utilitzar el mètode `addEventListener` de l'objecte `XMLHttpRequest` o les dreceres `onload`, `onerror` i `onprogress` per afegir la funció que serà cridada en detectar-se l'*event* corresponent.

> En la majoria dels casos és recomanable treballar amb els *events* `load`, `error` i `progress` en lloc de controlar els canvis d'estat.

Vegeu en l'exemple següent com el codi se simplifica molt:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script>
    if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+
          httpRequest = new XMLHttpRequest();
          console.log("Creat l'objecte a partir de XMLHttpRequest.");
        } else if (window.ActiveXObject) { // IE 6 i anteriors
          httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
          console.log("Creat l'objecte a partir d'ActiveXObject.");
        } else {
          console.error("Error: Aquest navegador no admet AJAX.");
        }

        httpRequest.onload = processarResposta;

        httpRequest.open('GET', 'provincies.txt', true);
        httpRequest.send(null);

        function processarResposta() {
          var resposta = httpRequest.responseText;
          var elementPre = document.createElement('pre');
          elementPre.innerHTML = resposta;
          document.body.appendChild(elementPre);
        }
  </script>
</head>

<body>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ZpVrZP?editors=1012.

Cal destacar que les etiquetes `provincies` i `provincia` no són visibles, com és d'esperar, ja que els navegadors no les representen. En canvi, sí que es pot visualitzar l'estructura del document fent servir les eines de desenvolupador per comprovar que s'han afegit correctament.

Un altre avantatge d'aquest sistema és que ofereix un mètode senzill per controlar el progrés de la petició, ja que mitjançant la detecció de l'*event* `progress` és possible controlar el progrés de la petició. Proveu d'afegir el següent codi a continuació de l'assignació d' `onload`:

```
httpRequest.onprogress = mostrarProgres;

```

```
function mostrarProgres(event) {
  if (event.lengthComputable) {
    var progres = 100 * event.loaded / event.total;
    console.log("Completat: " + progres + "%")
  } else {
    console.log("No es pot calcular el progrés");
  }
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/amPaYO?editors=1012.

En aquest exemple, com que es tracta d'una resposta molt curta, no es pot apreciar el progrés i automàticament retorna el 100%, però en cas d'enviament de formularis o pujada de fitxers al servidor es pot utilitzar per mostrar barres de progres a l'usuari.

1.3.2 Enviament de paràmetres

Habitualment, en fer una petició AJAX cal afegir-hi un o més paràmetres. Cal distingir entre dos tipus de formats per enviar els paràmetres:

- Cadena de consulta (*query string*): els paràmetres es passen codificats a l'URL, per exemple: `http://www.example.com?nom=Maria&cognom=Campmany`.
- Dades de formulari (*form data*): els paràmetres s'inclouen a la capçalera de la petició.

Quan el mètode d'enviament és `GET` s'han de codificar els paràmetres directament a la petició (és a dir, es tracta d'una *cadena de consulta*), de manera que en cridar el mètode `open` de l'objecte `XMLHttpRequest`, l'URL ja inclou aquests paràmetres.

Per exemple, serien paràmetres vàlids els que es mostren a l'exemple següent:

```
var httpRequest = new XMLHttpRequest();
var url = 'www.example.com';
var dades = {
  nom: 'Maria',
  cognom: 'Campmany Roig'
}

var params = convertirEnParametres(dades);

httpRequest.open('GET', url + "?" + params , true);

httpRequest.onload = function() {
  // Aquí es processaria la resposta de les peticions
};

httpRequest.send(null);

function convertirEnParametres(dades) {
  var params = '';
  for (var dada in dades) {
```

```
    if (params.length > 0) {
      params += '&'
    }

    params += encodeURIComponent(dada);
    params += '=';
    params += encodeURIComponent(dades[dada]);
  }

  return params;
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/zKgxqN?editors=0010.

Com es pot apreciar, s'ha creat una funció per convertir el diccionari de dades en una cadena de text amb el format adequat per afegir-la a l'URL. Fixeu-vos que s'ha utilitzat la funció `encodeURIComponent` per convertir les dades (tant la clau com el valor) en caràcters acceptats per la codificació de l'URL, com són els accents, les titlles i els espais.

Si examineu la capçalera de petició (o la pestanya "paràmetres" de la petició a Mozilla Firefox) amb les eines de desenvolupador del navegador, veureu que es reconeixen els paràmetres `nom` i `cognom` com a paràmetres de tipus cadena de consulta.

Podeu trobar més informació sobre les eines de desenvolupador al punt "Depuració de crides AJAX" d'aquest mateix apartat.

Cal destacar que tot i que `send` és l'encarregat de fer l'enviament i passar els paràmetres a la petició, en el cas del mètode d'enviament `GET` no és ben bé així, s'ha de codificar a l'URL.

Enviament de contingut binari mitjançant AJAX

Entre els tipus d'arguments que accepta el mètode `send` es troben `Blob` i `ArrayBufferView`, que s'utilitzen per enviar contingut binari, com per exemple una imatge.

En el cas d'altres mètodes com `POST`, `PUT/PATCH` i `DELETE` la petició s'ha de fer d'una manera diferent, i hi inclou modificar la capçalera de la petició per especificar el format en el qual s'envien les dades.

Per exemple, per afegir l'enviament de dades com a cadena de text fent servir el mètode `POST` no s'ha de modificar l'URL, però una vegada oberta la petició, cal establir a la capçalera el tipus de contingut com `application/x-www-form-urlencoded`:

```
var httpRequest = new XMLHttpRequest();
var url = 'www.example.com';
var dades = {
  nom: 'Maria',
  cognom: 'Campmany Roig'
}

var params = convertirEnParametres(dades);

httpRequest.open('POST', url, true);
httpRequest.setRequestHeader('Content-type', 'application/x-www-form-urlencoded
    ');

httpRequest.onload = function() {
  // Aquí es processaria la resposta de les peticions
};

httpRequest.send(params);

function convertirEnParametres(dades) {
  var params = '';
```

```
  for (var dada in dades) {
    if (params.length > 0) {
      params += '&'
    }

    params += encodeURIComponent(dada);
    params += '=';
    params += encodeURIComponent(dades[dada]);
  }

  return params;
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/VPXXqQ?editors=0010.

Fixeu-vos que només cal especificar que el mètode d'enviament serà `POST`; una vegada oberta la petició, s'ha d'especificar el tipus de contingut com a `application/x-www-form-urlencoded` i passar els paràmetres (convertits en una cadena codificada) com a argument al mètode `send`.

En alguns casos el servei web pot esperar rebre la informació en un altre format. Per fer-ho cal especificar aquest format a la capçalera i codificar les dades de la manera apropiada. Per exemple, si el servidor accepta els paràmetres en format JSON, el codi seria el següent:

```
var httpRequest = new XMLHttpRequest();
var url = 'www.example.com';
var dades = {
  nom: 'Maria',
  cognom: 'Campmany Roig'
}

httpRequest.open('POST', url, true);
httpRequest.setRequestHeader('Content-type', 'application/json');

httpRequest.onload = function() {
  // Aquí es processaria la resposta de les peticions
  var respostaJSON = JSON.parse(httpRequest.responseText);
};

httpRequest.send(JSON.stringify(dades));
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/KNwdWv?editors=0010.

Fixeu-vos que en aquest cas el tipus de contingut s'ha establert com a `application/json` i no és necessari codificar les dades perquè es pot convertir el diccionari de dades en una cadena de text amb format JSON mitjançant `JSON.stringify`. A la resposta s'ha afegit l'operació inversa, suposant que el retorn ha de ser una cadena de text també en format JSON, de manera que `respostaJSON` contindria un diccionari de dades amb la resposta.

Cal destacar que, utilitzant aquest sistema, el navegador Mozilla Firefox no detecta correctament els paràmetres; en canvi, Google Chrome ho mostra com a *Request Payload* i es poden visualitzar els paràmetres tant com a cadena de text en format JSON com a parells de dades clau-valor.

1.3.3 Processament de respostes com a HTML

Tot i que la interfície de `XMLHttpRequest` no incorpora cap mètode específic per recuperar la resposta en format HTML, és molt fàcil fer-ho directament a partir de la resposta textual.

Primerament creeu un nou fitxer anomenat `provincies.html` amb el codi següent:

```
<h1>Provincies</h1>
<ul>
  <li>Barcelona</li>
  <li>Girona</li>
  <li>Lleida</li>
  <li>Tarragona</li>
</ul>
```

Com que aquesta estructura no es correspon amb el format XML vàlid (hauria d'haver-hi només un element arrel) i en lloc d'enviar la petició al servidor s'està fent servir un fitxer local, és necessari sobreescriure el tipus multimèdia del fitxer de manera que el navegador pugui interpretar la resposta, ja que per defecte considera que totes les respostes són de tipus XML.

Per fer-ho s'ha d'invocar el mètode `overrideMimeType` i passar el tipus `text/html` com a argument, de manera que el codi per realitzar una petició simple quedaria així:

```
<html>
  <head>
    <meta charset="utf-8">

    <script>
      if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+
        httpRequest = new XMLHttpRequest();
        console.log("Creat l'objecte a partir de XMLHttpRequest.");
      } else if (window.ActiveXObject) { // IE 6 i anteriors
        httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        console.log("Creat l'objecte a partir d'ActiveXObject.");
      } else {
        console.error("Error: Aquest navegador no admet AJAX.");
      }

      httpRequest.onload = processarResposta;
      httpRequest.onprogress = mostrarProgres;

      function mostrarProgres(event) {
        if (event.lengthComputable) {
          var progres = 100 * event.loaded / event.total;
          console.log("Completat: " + progres + "%")
        } else {
          console.log("No es pot calcular el progrés");
        }
      }

      httpRequest.open('GET', 'provincies.html', true);
      httpRequest.overrideMimeType('text/html');
      httpRequest.send(null);

      function processarResposta() {
        var resposta = httpRequest.responseText;
```

```
34          var contenidor = document.createElement('div');
35          contenidor.innerHTML = resposta;
36          document.body.appendChild(contenidor);
37        }
38      </script>
39
40    </head>
41    <body>
42    </body>
43  </html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wzRLvB?editors=1012.

Gràcies a aquesta funcionalitat és possible enviar peticions al servidor per mostrar diàlegs que seguidament són renderitzats al client o fitxes de productes generades al servidor mitjançant plantilles (tot i que aquesta tasca es pot realitzar al client mitjançant entorns de treball com React, Ember o Angular).

1.3.4 Processament de respostes XML

XML

Podeu trobar més informació sobre XML ('llenguatge d'etiquetatge extensible') a l'enllaç següent: www.goo.gl/kBf1kz.

El format XML és un dels formats més utilitzats per intercanviar informació. Com que és el format pel qual, originalment, es transferia la informació utilitzant AJAX, l'objecte `XMLHttpRequest` disposa del mètode `responseXML` per obtenir la resposta directament en aquest format.

S'ha de tenir en compte que en cas que el servidor no especifiqui cap tipus multimèdia per obtenir la resposta (com passa amb els fitxers locals), el navegador pot esperar que es tracti de dades en format XML vàlid. Si no és així, llançarà una excepció i aturarà l'execució de l'aplicació.

Una resposta XML vàlida seria la següent:

```
1 <provincies>
2   <provincia cp="08">Barcelona</provincia>
3   <provincia cp="17">Girona</provincia>
4   <provincia cp="25">Lleida</provincia>
5   <provincia cp="43">Tarragona</provincia>
6 </provincies>
```

Perquè una resposta sigui considerada amb format XML vàlid ha de complir (entre d'altres) les condicions següents:

- Cada element ha d'estar format per una etiqueta d'apertura i una de tancament: .
- Un element pot incloure altres elements (l'element `provincies` conté 4 elements `provincia`).
- Tots els elements s'han de tancar en un ordre jeràrquic correcte, per exemple, no es pot tancar l'element `provincies` mentre algun dels elements `provincia` sigui obert.

- Ha d'existir un element arrel que engloba la resta d'elements del document (en aquest cas l'element arrel és `provincies`).
- Es poden especificar atributs assignant un valor entre cometes (`cp=‘‘08’’`).

Opcionalment, una resposta XML pot incloure un pròleg en el qual s'indiqui la versió XML i la codificació del document, entre d'altres. Per exemple, per indicar que correspon a la versió 1.0 d'XML i la codificació `ISO-859-1` (la corresponent a l'alfabet llatí, incloent-hi els diacrítics) al començament de la resposta es trobaria el codi següent:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Podeu trobar més informació sobre el DOM a la unitat “Programació amb el DOM” d'aquest mateix mòdul.

Cal recordar que les interfícies del DOM s'apliquen també als documents XML, de manera que es poden fer servir els mateixos mètodes o propietats per tractar tant HTML com XML, tal com podeu veure en l'exemple següent. Primerament, heu de crear un fitxer anomenat `provincies2.xml` amb el codi XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<provincies>
  <provincia cp="08">Barcelona</provincia>
  <provincia cp="17">Girona</provincia>
  <provincia cp="25">Lleida</provincia>
  <provincia cp="43">Tarragona</provincia>
</provincies>
```

Seguidament, creeu un fitxer amb el nom “processarXML.html” i enganxeu-hi el codi següent:

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <script>
    if (window.XMLHttpRequest) {
      httpRequest = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
      httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
    } else {
      console.error("Error: Aquest navegador no admet AJAX.");
    }

    httpRequest.onload = processarResposta;
    httpRequest.open('GET', 'provincies2.xml', true)
    httpRequest.send(null);

    function processarResposta() {
      var elementArrel = httpRequest.responseXML.documentElement;
      var elements = elementArrel.childNodes;
      var llista = document.createElement('ul');

      for (var i = 0; i < elements.length; i++) {
        if (elements[i].nodeType == Node.ELEMENT_NODE) {
          var item = processarElement(elements[i]);
          llista.appendChild(item);
        }
      }

      document.body.appendChild(llista);
    }

    function processarElement(element) {
      var codiPostal = element.getAttribute('cp');
```

```
34        var provincia = element.textContent;
35        var item = document.createElement('li');
36        item.textContent = provincia + ' (CP ' + codiPostal + ')';
37        return item;
38      }
39    </script>
40  </head>
41
42  <body>
43  </body>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/EgGBEZ?editors=1012.

Com es pot apreciar, s'ha descompost el processament de la resposta en dues funcions. Primer s'invoca la funció `processarResposta`, que obté el node arrel (que correspon a `provincies`) i a partir d'aquest obté la llista de nodes descendents (propietat `childNodes` de la interfície `Node` del DOM).

Cal destacar que la propietat `responseXML` conté un objecte que implementa la interfície `Document` del DOM i això permet obtenir l'element arrel a partir de la propietat `documentElement`.

Seguidament es recorre la llista dels nodes descendents de l'arrel i, una vegada s'ha comprovat que el tipus és `ELEMENT_NODE` (de manera que s'eviten els nodes de text), s'invoca la funció `processarElement` passant com a argument l'element. Aquesta funció extreu l'atribut `cp` i el contingut textual de l'element, crea un nou element de tipus `li` (element d'una llista d'HTML) i li assigna com a contingut una cadena composta pel nom de la província i el prefix del codi postal.

Els elements es processen un per un fins que no en queda cap, llavors s'afegeix la `llista` al cos del document.

Fixeu-vos que el tractament que es fa de la resposta XML és igual al que es pot fer a HTML, de manera que es possible extreure la informació dels atributs amb el mètode `getAttribute` i del contingut textual amb `textContent`.

S'ha de tenir en compte que en casos més complexos pot ser necessari utilitzar funcions recursives per recórrer completament el document. En aquest exemple s'ha considerat que l'arbre només constava de dos nivells i, per tant, només s'ha utilitzat un bucle per iterar sobre els elements. Per aquesta raó, en molts casos necessitareu implementar el vostre propi intèrpret (*parser* en anglès) que recorri tots els nodes iterativament i els processi un per un segons els requeriments de l'aplicació.

1.3.5 Processament de respostes JSON

Per processar una resposta en format JSON (JavaScript Object Notation) s'utilitza la propietat `responseText` de l'objecte `XMLHttpRequest` i seguidament s'analitza sintàcticament per convertir-la en un objecte de JavaScript, que es pot utilitzar com qualsevol altre.

Per comprovar-ho, creeu un fitxer de text anomenat `provincies.json` amb el contingut següent:

```
{
  "provincies": [
    {"nom": "Barcelona", "cp": "08"},
    {"nom": "Girona", "cp": "17"},
    {"nom": "Lleida", "cp": "25"},
    {"nom": "Tarragona", "cp": "43"}
  ]
}
```

L'estructura del format JSON és molt similar a la declaració literal d'objectes en JavaScript, ja que aquest és el seu origen. S'ha de tenir en compte que tant els noms de les propietats com els valors textuals han d'anar entre cometes dobles (no es poden utilitzar cometes simples). Encara que els nombres no és necessari que estiguin entre cometes, en aquest exemple s'han de tractar com cadenes de text per respectar el valor `08`.

El contingut de la resposta s'ha de trobar entre claus, que contenen una o més propietats amb un nom que es pot fer servir com a clau i un valor. Aquest valor pot ser qualsevol dels valors primitius de JavaScript com són els nombres i les cadenes de text, un *array* o un altre objecte literal. En el cas dels *arrays* el contingut ha d'anar entre claudàtors, `[` i `]`, mentre que els objectes han d'anar entre claus `{` i `}`.

Cal destacar que l'element arrel en una resposta en format JSON tant pot ser un objecte com un *array*.

El codi que s'ha de fer servir per realitzar la petició és pràcticament idèntic al de les peticions HTML: primer es crea l'objecte `XMLHttpRequest`, seguidament s'envia i finalment es processa la resposta:

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <script>
     if (window.XMLHttpRequest) {
       httpRequest = new XMLHttpRequest();
     } else if (window.ActiveXObject) {
       httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
     } else {
       console.error("Error: Aquest navegador no admet AJAX.");
     }

     httpRequest.onload = processarResposta;
     httpRequest.open('GET', 'provincies.json', true)
     httpRequest.overrideMimeType('text/plain');
     httpRequest.send(null);

     function processarResposta() {
       var resposta = JSON.parse(httpRequest.responseText);
       var llista = document.createElement('ul');

       for (var i = 0; i < resposta.provincies.length; i++) {
         var item = processarElement(resposta.provincies[i]);
         llista.appendChild(item);
       }

       document.body.appendChild(llista);
     }

     function processarElement(provincia) {
```

```
31        var item = document.createElement('li');
32        item.textContent = provincia.nom + ' (CP ' + provincia.cp + ')';
33        return item;
34      }
35    </script>
36  </head>
37
38  <body>
39  </body>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/qaLZyk?editors=1012.

Com podeu veure s'ha forçat el tipus multimèdia invocant el mètode `overrideMimeType` amb el valor `text/plain` per assegurar que es pot interpretar com a text sense que el navegador llanci cap excepció.

Un cop s'ha obtingut la resposta, s'analitza sintàcticament fent servir el mètode `parse` de l'objecte `JSON`, de manera que el resultat és un objecte JavaScript funcional que es pot utilitzar com un diccionari de dades.

Es recorren totes les províncies una per una i es retorna un element de tipus `li` amb el contingut textual corresponent a la província especificada com a argument.

Fixeu-vos que la implementació en aquest cas és més simple, ja que la conversió de la resposta en un objecte de JavaScript és directa i, per tant, es pot accedir a les seves propietats fent servir la notació de punt (`provincia.nom`) o la notació de claudàtors (`provincia['nom']`).

Com en el cas del processament de respostes XML, s'ha de tenir en compte que en determinats casos (quan no coneixem la resposta a priori, per exemple) pot ser necessari implementar una funció recursiva per recórrer tots els elements de l'arbre per processar-los.

La utilització d'un format o un altre dependrà del format en el qual el servidor retorna la resposta, tot i que en alguns casos s'ofereix accedir a la mateixa informació tant en XML com en JSON. En aquests casos pot ser més fàcil treballar amb el format JSON, perquè es pot manipular la resposta directament sense necessitat d'utilitzar les interfícies del DOM.

1.4 Serveis web

En moltes ocasions és necessari implementar un sistema informàtic que faciliti l'intercanvi d'informació entre diferents plataformes, així és possible accedir a les mateixes dades des d'una pàgina web o una aplicació mòbil canviant només la interfície en la qual es mostren.

La implementació d'aquests serveis web depèn del servidor i és el servidor el que determina la forma d'accedir a aquestes dades. Les dues arquitectures més utilitzades són SOAP (Simple Object Access Protocol) i REST (Representational

State Transfer); per conseqüent, cal conèixer-les per poder desenvolupar aplicacions web que connectin amb aquests serveis, mitjançant AJAX o bé CORS (Cross-Origin Resource Sharing).

Cal destacar que la implementació del servidor és independent de la implementació dels clients, ja que el servidor permet connectar amb l'aplicació sense que importi la plataforma o el sistema operatiu que s'hagin fet servir. Només cal que els clients implementin correctament els mecanismes de connexió.

Un avantatge dels serveis web és que quan es treballa sobre el protocol HTTP, per accedir-hi no cal fer cap modificació als clients, al contrari d'altres tipus d'aplicacions que requereixen utilitzar ports específics i que poden ser bloquejats pels tallafocs, tant de l'equip com de l'encaminador a través del qual es connecten a internet.

1.4.1 SOAP

SOAP

Podeu trobar més informació sobre el protocol SOAP ('protocol d'accés a objecte simple') a l'enllaç següent: www.goo.gl/Zufz8w.

WSDL

Podeu trobar més informació sobre WSDL ('llenguatge de descripció de serveis web') a l'enllaç següent: www.goo.gl/SU5j3d.

SOAP és un protocol utilitzat per intercanviar informació en una xarxa d'ordinadors. Utilitza el format XML tant per negociar la connexió com per generar la resposta i pot funcionar sobre diferents protocols de xarxa com HTTP, SMTP, TCP o UDP.

Tot i que no és imprescindible, es pot utilitzar un fitxer WSDL (Web Services Description Language) per definir la descripció del *web service*. Aquest fitxer, en format XML, conté la informació necessària perquè els clients puguin conèixer i utilitzar les operacions que ofereix el servei web.

Entre els desavantatges d'aquesta arquitectura es troba la llargària dels missatges, la complexitat d'implementar els serveis, tant a la banda del client com a la del servidor, i la lentitud en analitzar sintàcticament les respostes XML.

1.4.2 REST

REST

Podeu trobar més informació sobre l'arquitectura REST ('transferència d'estat de representació') a l'enllaç següent: www.goo.gl/bUr2TO.

Mentre que SOAP és un protocol i disposa d'una interfície estàndard, REST és un estil d'arquitectura i no s'aplica cap estàndard a si mateix, tot i que sí que en fa servir alguns com HTTP, URI, JSON i XML.

Els serveis web REST són més simples d'implementar que els serveis SOAP i força més lleugers. Per accedir-hi només cal conèixer l'URI (*uniform resource identifier*, 'identificador uniforme de recursos') del punt d'accés per l'acció (*endpoint*) i el verb o mètode que s'ha d'utilitzar.

Els verbs es corresponen amb els mètodes del protocol HTTP són els següents:

- **GET**: per obtenir dades.

- **PUT**: per reemplaçar dades.
- **PATCH**: per actualizar dades.
- **POST**: per afegir una nova dada.
- **DELETE**: per eliminar una dada.

D'aquesta manera, a partir d'un mateix URL, segons el verb emprat el resultat serà diferent. Per exemple, a partir de l'URL (fictícia) `http://api.example.com/recursos/`:

- **GET**: obtindria una llista de tots els recursos.
- **PUT**: reemplaçaria un recurs existent amb les dades enviades com a paràmetres (des d'un formulari, per exemple).
- **PATCH**: actualitzaria les dades d'un recurs existent.
- **POST**: crearia un nou recurs a partir de les dades enviades com a paràmetres.
- **DELETE**: eliminaria la col·lecció `recursos`.

En aquesta arquitectura les accions realitzades utilitzant el mètode `GET` no han de provocar cap modificació a les dades i, en conseqüència, es tracta d'un mètode segur. Per contra, els altres tres sempre les modifiquen.

Un avantatge important sobre SOAP és que no requereix utilitzar XML, així que per l'intercanvi de dades es pot utilitzar JSON o altres formats de representació que poden resultar més pràctics i fàcils de comprovar.

A més a més, el protocol REST és molt lleuger i quan es fan servir altres formats de transferència com JSON, poden minimitzar el consum d'amplada de banda tant per al servidor com per al client.

A diferència de SOAP, els serveis web REST funcionen només sobre el protocol HTTP, ja que depenen de la utilització dels seus mètodes per determinar quina acció s'ha de portar a terme sobre les dades.

S'ha de tenir en compte que tant el format com l'estructura de les dades que s'intercanvien són lliures; així doncs, caldrà comprovar la documentació del servei o analitzar sintàcticament la resposta per poder interpretar-les correctament. Per exemple, és possible que la resposta arribi en format JSON amb dos camps, un amb el codi de la resposta i un altre amb les dades:

```
{
  "codi": 200
  "dades": {
    provincies: ["Barcelona", "Girona", "Lleida", "Tarragona"]
  }
}
```

En altres casos podria arribar només la informació sense cap codi, ja que es pot extreure del codi d'estat de la capçalera (*propietat state*), llavors la resposta podria ser similar a aquesta:

```
{
  provincies: ["Barcelona", "Girona", "Lleida", "Tarragona"]
}
```

Cal destacar que amb aquesta arquitectura és molt fàcil comprovar-ne el funcionament, sobretot amb el mètode `GET`, ja que només s'ha d'introduir al navegador l'URI del punt d'accés amb els paràmetres (amb aquesta forma: `www.example.com/recursos?asignatura=m06&unitat=2`) i es carregarà el resultat al mateix navegador.

1.5 JSONP i CORS

A causa de les limitacions imposades per la política del mateix domini no es poden fer peticions AJAX a dominis diferents del que es troba l'aplicació. Per superar aquesta limitació hi ha dues tècniques alternatives: JSONP (JSON amb *padding*) i CORS (Cross-Origin Resource Sharing).

Malauradament, totes dues requereixen que el servidor estigui preparat per acceptar aquestes peticions de dominis externs i, per consegüent, no totes les fonts de dades són accessibles des de aplicacions web encara que ofereixin una API (interfície de programació d'aplicacions) que exposi aquestes dades.

S'ha de tenir en compte que la política del mateix domini només afecta els navegadors i, per tant, els serveis web que no admeten ni JSONP ni CORS continuen sent accessibles per aplicacions d'escriptori o mòbils.

1.5.1 JSON amb padding (JSONP)

La tècnica del JSON amb *padding* consisteix a carregar la informació com si es tractés d'un script que inclou una invocació a una funció amb totes les dades com a paràmetre (el *padding* es refereix a això).

Aquesta tècnica funciona perquè la càrrega de scripts no està subjecta a la política del mateix origen i permet carregar scripts que invoquin automàticament altres funcions.

API de Flickr

Podeu trobar tota la informació per desenvolupadors de l'API de Flickr a l'enllaç següent: www.goo.gl/AmwqQl.

Per exemple, la resposta obtinguda en carregar l'URL bit.ly/2ocrAKQ incrusta un script a la pàgina amb una invocació a la funció `processar`, que és el valor del paràmetre `jsoncallback`, i que utilitza el servei web de Flickr (www.flickr.com) per generar la resposta JSONP i li passa totes les dades en format JSON com a argument:

```
processar({
```

```
    "title": "Recent Uploads tagged kitten",
    "link": "http://www.flickr.com/photos/tags/kitten/",
    "description": "",
    "modified": "2016-10-22T18:01:41Z",
    "generator": "http://www.flickr.com/",
    "items": [
     {
      "title": "Cats images Beautiful Eyes via http://ift.tt/29KELz0",
      "link": "http://www.flickr.com/photos/dozhub/29860948003/",
      "media": {"m":"http://farm9.staticflickr.com/8677/29860948003
          _31626a7d77_m.jpg"},
      "date_taken": "2016-10-22T11:01:41-08:00",
      "description": " <p><a href=\"http://www.flickr.com/people/dozhub/\">
          dozhub<\/a> posted a photo:<\/p> <p><a href=\"http://www.flickr.com/
          photos/dozhub/29860948003/\" title=\"Cats images Beautiful Eyes via
          http://ift.tt/29KELz0\"><img src=\"http://farm9.staticflickr.com
          /8677/29860948003_31626a7d77_m.jpg\" width=\"240\" height=\"160\"
          alt=\"Cats images Beautiful Eyes via http://ift.tt/29KELz0\" /><\/a
          ><\/p> <p>Cats images Beautiful Eyes -<br /> <br /> Cats images
          Beautiful Eyes - Cats, kittens and kittys, cute and adorable! Aww! (
          via <a href=\"http://ift.tt/29KELz0\" rel=\"nofollow\">ift.tt/29
          KELz0<\/a>)<br /> <br /> - via <a href=\"http://ift.tt/29KELz0\" rel
          =\"nofollow\">ift.tt/29KELz0<\/a>. Cats, kittens and kittys, cute
          and adorable! Aww!<\/p>",
      "published": "2016-10-22T18:01:41Z",
      "author": "nobody@flickr.com (dozhub)",
      "author_id": "143919671@N07",
      "tags": "cat kitty kitten cute funny aww adorable cats"
     }]
  });
```

Així doncs, una vegada es carrega l'script s'executa la funció `processar`, que prèviament haurem definit i rep com a paràmetre totes les dades per processar-les.

Com es pot apreciar, només es pot utilitzar aquesta tècnica si el servidor està preparat per servir les dades en aquest format.

Vegeu a continuació un exemple que extreu la informació de l'agregador (*feed*) de Flickr per mostrar fotos de gatets:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>
</body>
<script>
  function processar (dades) {
    var imatges = dades.items;

    for (var i=0; i<imatges.length; i++) {
      var img = document.createElement('img');
      img.setAttribute('src', imatges[i].media.m);
      img.setAttribute('title', imatges[i].title);
      img.setAttribute('alt', imatges[i].title);
      document.body.appendChild(img);
    }
  };
</script>

<script src='http://api.flickr.com/services/feeds/photos_public.gne?
    jsoncallback=processar&tags=kitten&format=json'></script>
</script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/NRoWwR?editors=1000.

Fixeu-vos que a l'exemple a CodePen s'ha fet servir només la pestanya de codi HTML. Això s'ha fet així per assegurar que primer es defineix la funció `processar` i seguidament es realitza la petició JSONP. En cas contrari, la resposta de la petició pot arribar abans de definir-se la funció i produir-se un error, ja que la funció a executar encara no es troba definida.

La implementació de la funció `processar` és molt simple: com que la informació ha arribat en format JSON, s'ha convertit automàticament en un objecte JavaScript sense haver d'analitzar-la sintàcticament, així que només cal iterar sobre les dades i crear un element de tipus `img` per a cada imatge establint els atributs `src`, `title` i `imatge` associats.

1.5.2 Cross-Origin Resource Sharing (CORS)

CORS és un mecanisme que permet als navegadors moderns realitzar peticions AJAX a dominis diferents del que es troba l'aplicació web, utilitzant l'objecte `XMLHttpRequest` però assegurant la seguretat de la transmissió de dades en col·laboració amb el servidor.

CORS

Podeu trobar més informació sobre CORS ('control d'accés HTTP') a l'enllaç següent: www.goo.gl/CHQXLa.

En enviar una petició, hi ha un intercanvi de dades entre el client i el servidor per comprovar si la transmissió és acceptable o no. Aquestes comprovacions poden incloure diferents factors com l'origen de la petició o l'acreditació de l'usuari.

Si el servidor envia a la capçalera el paràmetre `Access-Control-Allow-Origin` amb un valor que correspongui al del domini de l'aplicació o `*`, la transmissió es pot dur a terme i la resposta contindrà les dades demanades. En altres casos és necessari comprovar l'acreditació i es necessitarà establir la propietat `withCredentials=true` de l'objecte que envia la petició per habilitar l'enviament de les galetes al servidor, on es comprovarà la validesa de les credencials.

> El mecanisme CORS és **transparent** al codi del client, ja que és gestionat per la implementació del servidor i dels navegadors.

A continuació, podeu veure un exemple en què s'envia una petició a una de les fonts de dades obertes de l'Ajuntament de Barcelona i es recupera la llista de festivals de l'any 2015 en format JSON:

```
<!DOCTYPE html>

<head>
  <meta charset="utf-8">
  <script>
    if (window.XMLHttpRequest) {
      httpRequest = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
      httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
    } else {
```

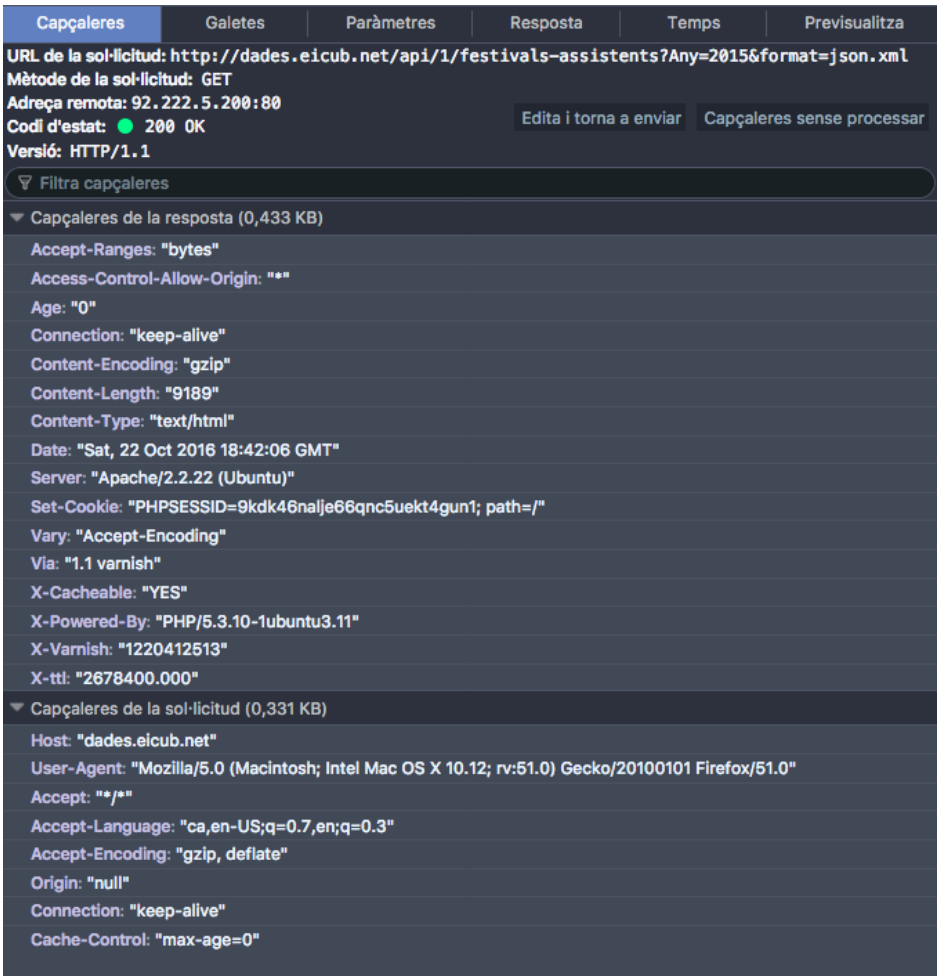
```
      console.error("Error: Aquest navegador no admet AJAX.");
    }

    httpRequest.onload = processarResposta;
    httpRequest.open('GET', 'http://dades.eicub.net/api/1/festivals_assistents?
        Any=2015&format=json.xml', true)
    httpRequest.overrideMimeType('text/plain');
    httpRequest.send(null);

    function processarResposta() {
      var resposta = JSON.parse(httpRequest.responseText);
      var llista = document.createElement('ul');
      console.log(resposta);

      for (var i = 0; i < resposta.length; i++) {

          var item = processarDada(resposta[i]);
          llista.appendChild(item);
      }

      document.body.appendChild(llista);
    }

    function processarDada(dada) {
      var item = document.createElement('li');
      var enllac = document.createElement('a');
      enllac.textContent = dada.NomDelFestival;
      if (dada.Web) {
        enllac.setAttribute('href', dada.Web);
      }
      enllac.setAttribute('title', dada.Organitzador);
      item.appendChild(enllac);

      return item;
    }
  </script>
</head>

<body>
  <h1>Festivals 2015</h1>
</body>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ALNjyG?editors=1010.

Si examineu la capçalera de la petició amb les eines de desenvolupador del navegador, veureu que inclou el paràmetre `Access-Control-Allow-Origin` amb valor `*`. És a dir, admet connexions des de qualsevol domini, com es pot apreciar a la figura 1.1.

CORS inclou altres mecanismes més complexos com són les restriccions de les peticions segons els mètodes empleats, l'ús de credencials o la modificació de les capçaleres, però tots treballen en conjunció amb el servidor i, consegüentment, si no es té accés al servidor o aquest no ha estat preparat per treballar amb aquests mecanismes no es podrà accedir a les dades des d'altres dominis.

FIGURA 1.1. Capçaleres d'una petició CORS



1.6 Accés a dades obertes

Les dades obertes són conjunts de dades posades a disposició del públic i que es poden fer servir sense restriccions. Habitualment aquestes dades poden ser consultades al mateix lloc web, descarregades o consumides per altres serveis.

Aquesta darrera opció permet el desenvolupament d'aplicacions que utilitzin dades de tercers; per exemple, per mostrar els pròxims esdeveniments a una ciutat o l'estat del trànsit.

Un conjunt de dades, perquè es considerin obertes, han de poder ser reutilitzables, adaptables i distribuïbles pels usuaris.

S'ha de tenir en compte que per poder accedir a aquestes dades els serveis web que les exposen han d'admetre la utilització de JSONP o de CORS, ja que en cas contrari seran bloquejats per la política del mateix origen dels navegadors.

En aquest aspecte, l'Ajuntament de Barcelona ofereix una gran selecció de dades obertes a les quals es pot accedir des de l'adreça següent: www.opendata.bcn.cat/opendata. Malauradament, la majoria de dades d'aquest

catàleg no admeten ni JSONP ni CORS i, per conseqüentment, no es poden accedir des d'aplicacions web.

Per comprovar si una font de dades admet JSONP s'ha de consultar la documentació de l'API del servei web que es vol utilitzar. Aquesta documentació pot ser disponible o no; per exemple, al catàleg de dades obertes de l'Ajuntament de Barcelona no es pot trobar en la majoria de les fonts. Per contra, les dades ofertes per l'Observatori de dades culturals de Barcelona (que forma part del mateix Ajuntament) sí que ofereix documentació per accedir a les seves dades, que es poden trobar a l'enllaç següent: www.barcelonadadescultura.bcn.cat/dades-obertes.

Comprovar si un lloc admet la transmissió de dades mitjançant el mecanisme CORS bàsic (sense cap requisit) és molt més simple: només cal consultar a les eines de desenvolupador la capçalera de la resposta en accedir directament a l'adreça. Si la capçalera inclou el paràmetre `Access-Control-Allow-Origin=''*''` (l'asterisc vol dir que accepta qualsevol origen), voldrà dir que s'hi pot accedir utilitzant AJAX directament, perquè el mecanisme CORS està habilitat al servidor.

Per exemple, si accediu a www.dades.eicub.net/api/1/museusexposicions-visitants (que forma part del catàleg de dades obertes de l'Observatori de dades culturals de Barcelona) i a les eines de desenvolupador seleccioneu la pestanya per inspeccionar el tràfic a la xarxa (pestanya *Network* tant a Google Chrome com a Mozilla Firefox), en fer clic sobre la petició, podeu comprovar que, efectivament, a la capçalera es troba el paràmetre `Access-Control-Allow-Origin` amb valor `*` i, per tant, es poden realitzar peticions des de diferents dominis.

Podeu comprovar-ho amb l'exemple següent, que realitza una petició al catàleg de dades, processa aquestes dades afegint-les a un diccionari de dades propi i les incorpora a una llista desplegable que, en seleccionar l'identificador, mostra les dades enllaçades:

```
<head>
  <meta charset="utf-8">

  <style>
    h1 {
      text-align:center;
    }

    fieldset {
      width: 300px;
      margin: 0 auto;
    }
    label {
      display: inline-block;
      width: 150px;
      font-weight: bold;
    }
    span {
      display: inline-block;
      width: 150px;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
  </style>
```

```
27
28  </head>
29
30  <body>
31    <h1>Activitats de difusió de les fàbriques de creació</h1>
32    <fieldset>
33      <select id="identificador">
34      <option>Selecciona un identificador</option></select>
35      <ul>
36        <li><label>Id:</label><span id="id"></span></li>
37        <li><label>Any:</label><span id="any"></span></li>
38        <li><label>Equipament:</label><span id="equipament"></span></li>
39        <li><label>Districte:</label><span id="districte"></span></li>
40        <li><label>Àmbit:</label><span id="ambit"></span></li>
41        <li><label>Assistents:</label><span id="asistents"></span></li>
42        <li><label>Notes:</label><span id="notes"></span></li>
43        <li><label>Tipus d'equipament:</label><span id="tipusEquipament"></span><
              /li>
44
45        <li><label>Titularitat:</label><span id="titularitat"></span></li>
46        <li><label>Activitats de difusió dins dels centres:</label><span id="
              activitats"></span></li>
47      </ul>
48    </fieldset>
49
50  <script>
51    if (window.XMLHttpRequest) {
52      httpRequest = new XMLHttpRequest();
53    } else if (window.ActiveXObject) {
54      httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
55    } else {
56      console.error("Error: Aquest navegador no admet AJAX.");
57    }
58
59    var dades = {}; // Diccionari on es guardaran les dades
60
61    httpRequest.onload = processarResposta;
62    httpRequest.open('GET', 'http://dades.eicub.net/api/1/fabriquescreacio-
          difusio', true)
63    httpRequest.overrideMimeType('text/plain');
64    httpRequest.send(null);
65
66    function processarResposta() {
67      var resposta = JSON.parse(httpRequest.responseText);
68      var llistaDesplegable = document.getElementById('identificador');
69
70      for (var i = 0; i < resposta.length; i++) {
71        var item = processarDada(resposta[i]);
72        dades[resposta[i].Id] = resposta[i];
73        llistaDesplegable.appendChild(item);
74      }
75
76      llistaDesplegable.onchange = actualitzarDadesMostrades;
77
78    }
79
80    function processarDada(dada) {
81      var item = document.createElement('option');
82      item.setAttribute('value', dada.Id);
83      item.textContent = dada.Id;
84      return item;
85    }
86
87    function actualitzarDadesMostrades(event) {
88        var llistaDesplegable = document.getElementById('identificador');
89      var dada = dades[llistaDesplegable.value]
90
91      console.log(dada);
92      actualitzarDadaMostrada('id', dada.Id);
93      actualitzarDadaMostrada('any', dada.Any);
```

```
94      actualitzarDadaMostrada('equipament', dada.Equipament);
95      actualitzarDadaMostrada('districte', dada.Districte);
96      actualitzarDadaMostrada('ambit', dada.Ambit);
97      actualitzarDadaMostrada('asistents', dada.Assistents);
98      actualitzarDadaMostrada('notes', dada.Notes || 'No hi ha cap nota');
99      actualitzarDadaMostrada('tipusEquipament', dada.TipusEquipament);
100     actualitzarDadaMostrada('titularitat', dada.Titularitat);
101     actualitzarDadaMostrada('activitats', dada.
            Activitats_de_difusio_dins_dels_centres);
102
103   }
104
105   function actualitzarDadaMostrada(nom, valor) {
106     document.getElementById(nom).textContent = valor;
107   }
108
109 </script>
110 </body>
```

Podeu veure aquest exemple en l'enllaç següent: www.http://codepen.io/ioc-daw-m06/pen/jrdVNd?editors=1111.

Com es pot apreciar, com que es tracta d'un servei web que admet CORS, només cal fer una petició AJAX convencional i processar les dades.

Cal tenir en compte que una mateixa pàgina pot fer peticions a múltiples fonts de dades. Per exemple, seria possible fer consultes a un catàleg de dades d'esdeveniments culturals on s'incloguessin les coordenades de geolocalització i utilitzar aquestes dades per afegir marques a un mapa de Google Maps (que fa servir les seves pròpies crides AJAX).

1.7 API Web Sockets

En determinades situacions, la transmissió de dades mitjançant AJAX no és viable, per exemple, en el cas de jocs multijugador en temps real o les aplicacions de missatgeria.

Cal recordar que les peticions al servidor fent servir el protocol HTTP no conserven l'estat, sinó que cada petició es considera nova i s'han d'utilitzar altres mecanismes com les galetes i l'enviament de paràmetres perquè el servidor reconegui l'usuari.

La solució és fer servir altres protocols com TCP i UDP, que permeten mantenir la comunicació oberta entre el client i el servidor per enviar i rebre missatges.

Web Sockets

Podeu trobar més informació sobre l'API Web Sockets i l'objecte `WebSocket` als enllaços següents, respectivament: www.goo.gl/eJgu0q i www.goo.gl/OF78BK

Per implementar aquesta funcionalitat a JavaScript s'ha d'utilitzar l'API Web Sockets, introduïda a HTML5, que permet realitzar connexions a servidors utilitzant internament el protocol TCP. Els protocols utilitzats per connectar amb el servidor són `ws` o `wss` per a sòcols segurs.

Mentre que fent servir peticions AJAX s'han de fer peticions periòdiques al servidor per comprovar si hi ha noves dades, això no cal fer-ho quan s'utilitzen Web Sockets, perquè una vegada hi ha dades noves al servidor, s'envien al client, i així s'estalvien recursos i amplada de banda tant al client com al servidor.

S'ha de tenir en compte que aquesta tecnologia és experimental, així doncs, abans de començar la implementació de qualsevol aplicació s'ha de consultar documentació actualitzada, ja que és possible que canviïn els noms de propietats, mètodes o els seus paràmetres.

Per descomptat, per poder realitzar una connexió amb un servidor, ha d'acceptar connexions TCP. Cada llenguatge de programació ofereix les seves alternatives; així doncs, es poden trobar servidors per Web Sockets en pràcticament qualsevol llenguatge.

Tot i així, s'ha de tenir en compte que no tots els llenguatges tenen el mateix nivell de compatibilitat en aquest àmbit, de manera que si voleu implementar un servidor de Web Sockets en PHP, el nombre de fonts que caldrà consultar serà molt menor que si ho feu amb Node.js (JavaScript a la banda del servidor), que té un grau de compatibilitat molt superior i API especialitzades com Socket.IO (www.socket.io) per simplificar aquestes tasques.

A l'hora de desenvolupar aplicacions segures utilitzant Web Sockets s'ha de tenir en compte que els navegadors no admeten determinats comportaments. Per exemple, si la pàgina on s'executa l'aplicació utilitza el protocol HTTP, l'aplicació pot fer servir el protocol `ws` o `wss`. Per contra, si la pàgina fa servir el protocol HTTPS (HTTP segur), s'haurà d'utilitzar forçosament el protocol `wss` o la connexió serà bloquejada pel navegador.

Per altra banda, tampoc no està permesa la utilització de certificats (SSL o TLS) autosignats, sinó que han d'utilitzar-se certificats aprovats per una autoritat de certificació i lligats a un nom de domini.

1.8 Depuració de crides AJAX

Per desenvolupar aplicacions és imprescindible saber les eines de depuració que teniu al vostre abast. Afortunadament, les eines més importants de les que es disposa a l'hora de depurar una aplicació web estan incorporades en els navegadors més populars, com Google Chrome i Mozilla Firefox.

Concretament, aquests dos navegadors fan servir interfícies d'usuari molt similars, com es pot apreciar a la figura 1.2, corresponent a les eines de xarxa de Google Chrome, i la figura 1.3, que mostra les de Mozilla Firefox.

Com podeu veure en els dos casos, la pestanya que conté la informació sobre les peticions s'anomena `xarxa` (*network* en anglès). En tots dos casos es disposa d'una barra superior on es poden filtrar les peticions que cal visualitzar: XHR, JS, CSS, Img, WS, etc. Això facilita localitzar ràpidament el recurs que volem consultar, ja que una pàgina pot realitzar desenes de peticions.

FIGURA 1.2. Eines de desenvolupador a Google Chrome

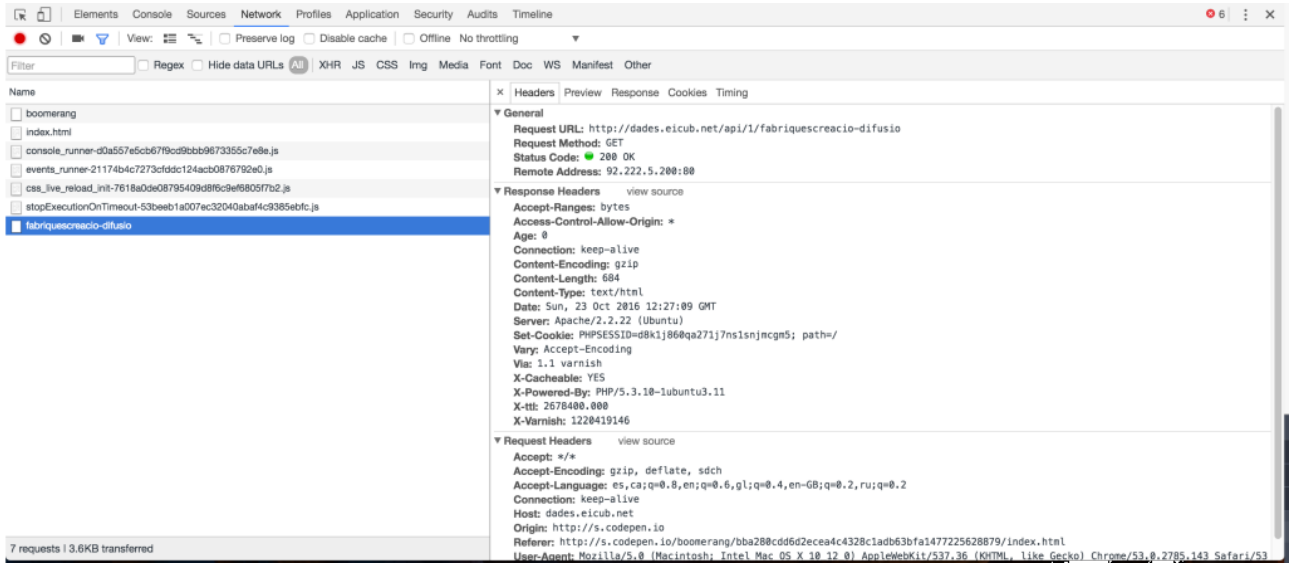
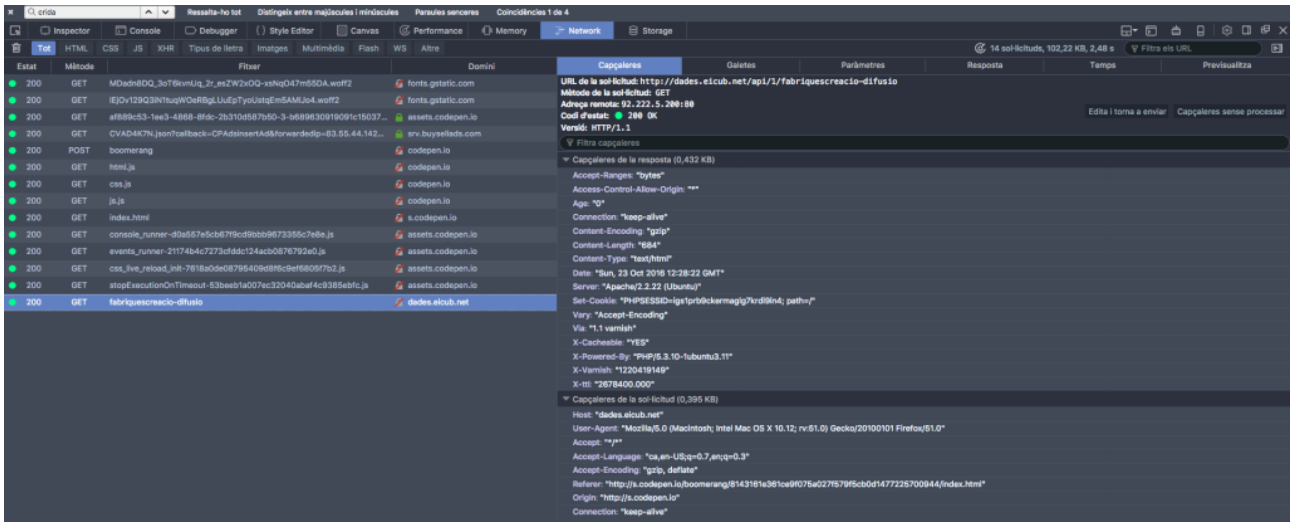


FIGURA 1.3. Eines de desenvolupador a Mozilla Firefox



Una vegada es clica sobre una de les peticions, la pestanya es divideix en dues zones: a l'esquerra es continua veient la informació general de les peticions, i a la dreta la informació concreta, dividida al seu torn en cinc pestanyes (capçaleres, previsualització, resposta, galetes i temps).

Per comprovar les dades de la petició i la resposta podeu fer servir el codi següent, que envia una petició a un servei web que admet peticions de diferents dominis:

```
1  <html>
2  <head>
3    <meta charset="utf-8">
4    <script>
5    httpRequest = new XMLHttpRequest();
6
7    httpRequest.onload = processarResposta;
8    httpRequest.open('GET', 'http://dades.eicub.net/api/1/fabriquescreacio-
         difusio', true)
9    httpRequest.overrideMimeType('text/plain');
10   httpRequest.send(null);
11
12   function processarResposta() {
13     console.log("Dades carregades correctament");
```

```
14    }
15    </script>
16  </head>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wzNgrB?editors=1011.

A la pestanya de la capçalera es pot veure tant la informació d'enviament com la resposta del servidor. Entre la informació que es mostra en aquesta pestanya cal destacar l'URL al qual s'ha enviat la petició, el mètode, el codi d'estatus (200 per a una petició realitzada amb èxit, 404 per a una petició no trobada, 500 per a un error intern del servidor, etc.) i en el cas de peticions a altres dominis, la presència del paràmetre `Access-Control-Allow-Origin` si el servidor admet CORS.

Cal destacar que Mozilla Firefox inclou a aquesta pestanya el botó *Edita i torna a enviar*, que permet editar manualment les dades de la capçalera de la petició i reenviar-la.

Podeu veure a la figura 1.4 la mostra d'una capçalera en Google Chrome i a la figura 1.5 la mateixa capçalera en Mozilla Firefox.

FIGURA 1.4. Capçaleres d'una petició a Google Chrome

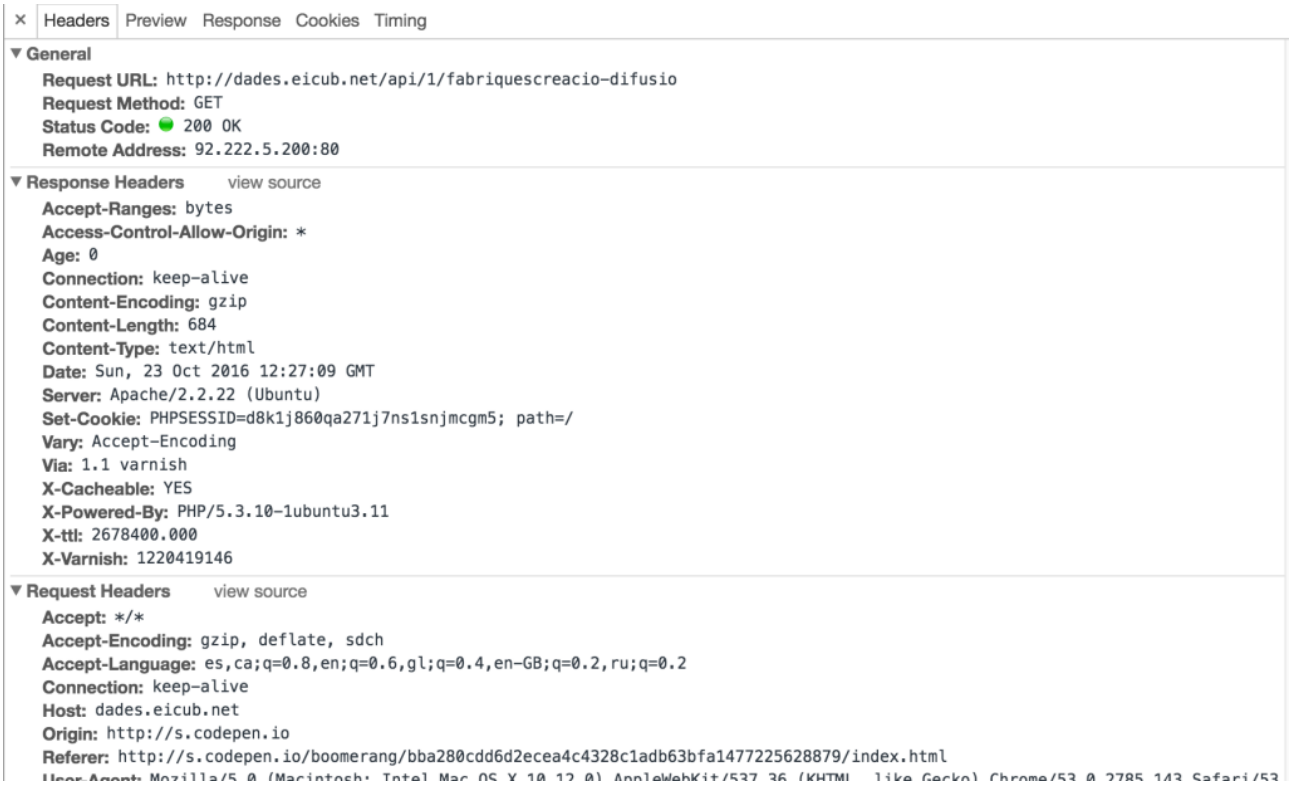
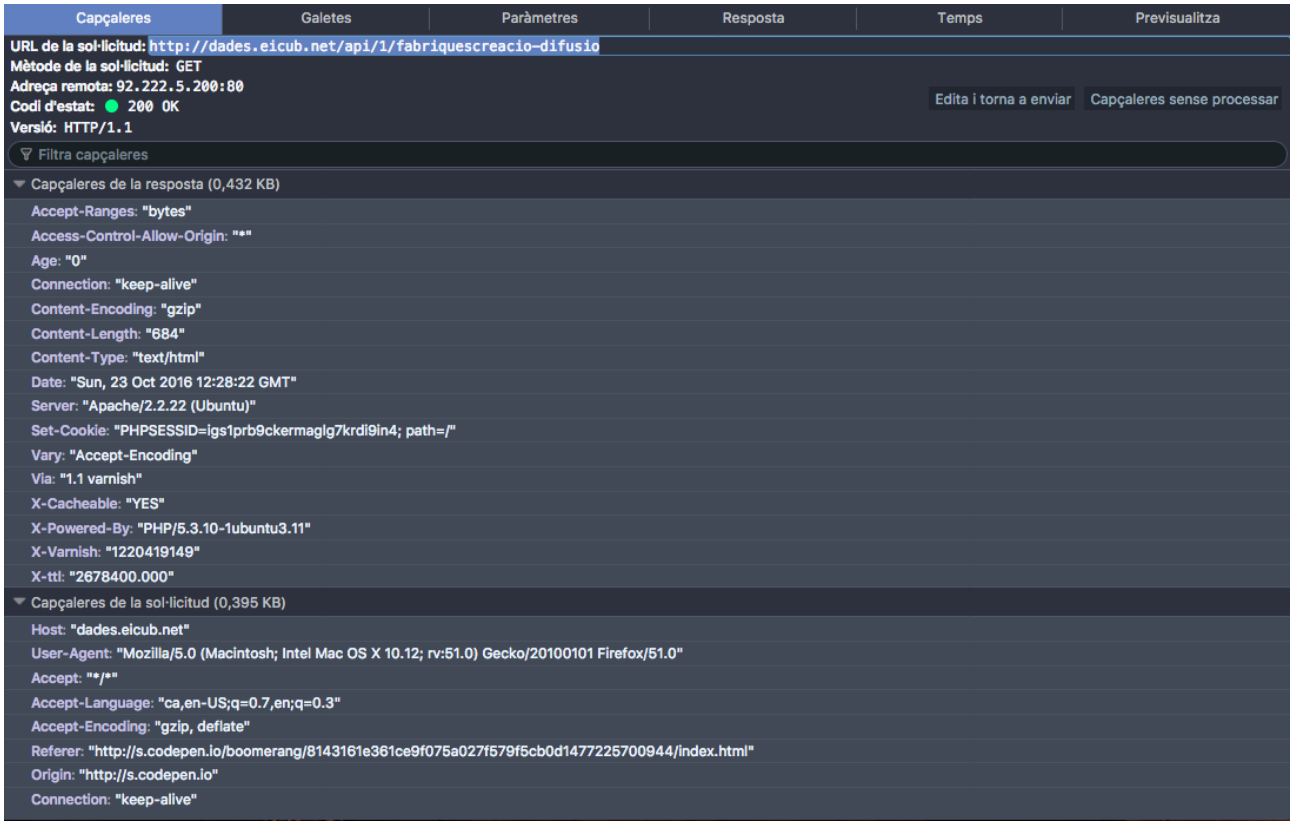


FIGURA 1.5. Capçaleres d'una petició a Mozilla Firefox



La pestanya de previsualització és la que mostra més diferències entre els dos navegadors: mentre que a Google Chrome es mostra la informació correctament estructurada com a objectes de JavaScript (com es pot apreciar a la figura 1.6), a Mozilla Firefox es mostra com a text pla (vegeu la figura 1.7). Cal destacar que en el cas del codi HTML el més habitual és que es mostri una previsualització de la pàgina.

FIGURA 1.6. Previsualització d'una petició a Google Chrome

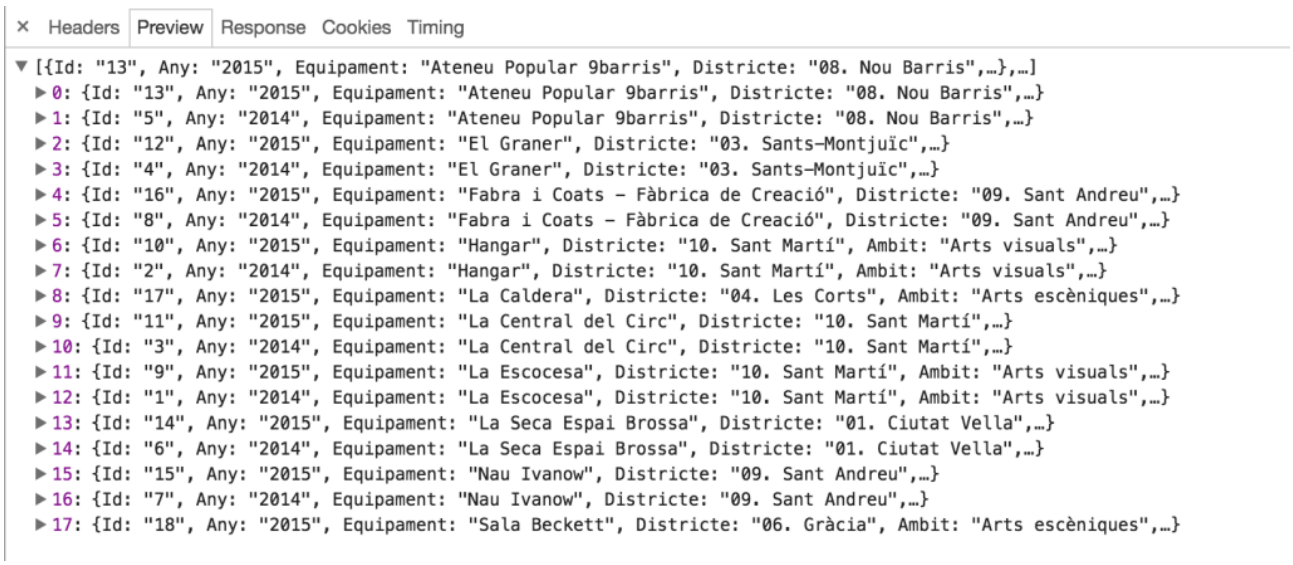
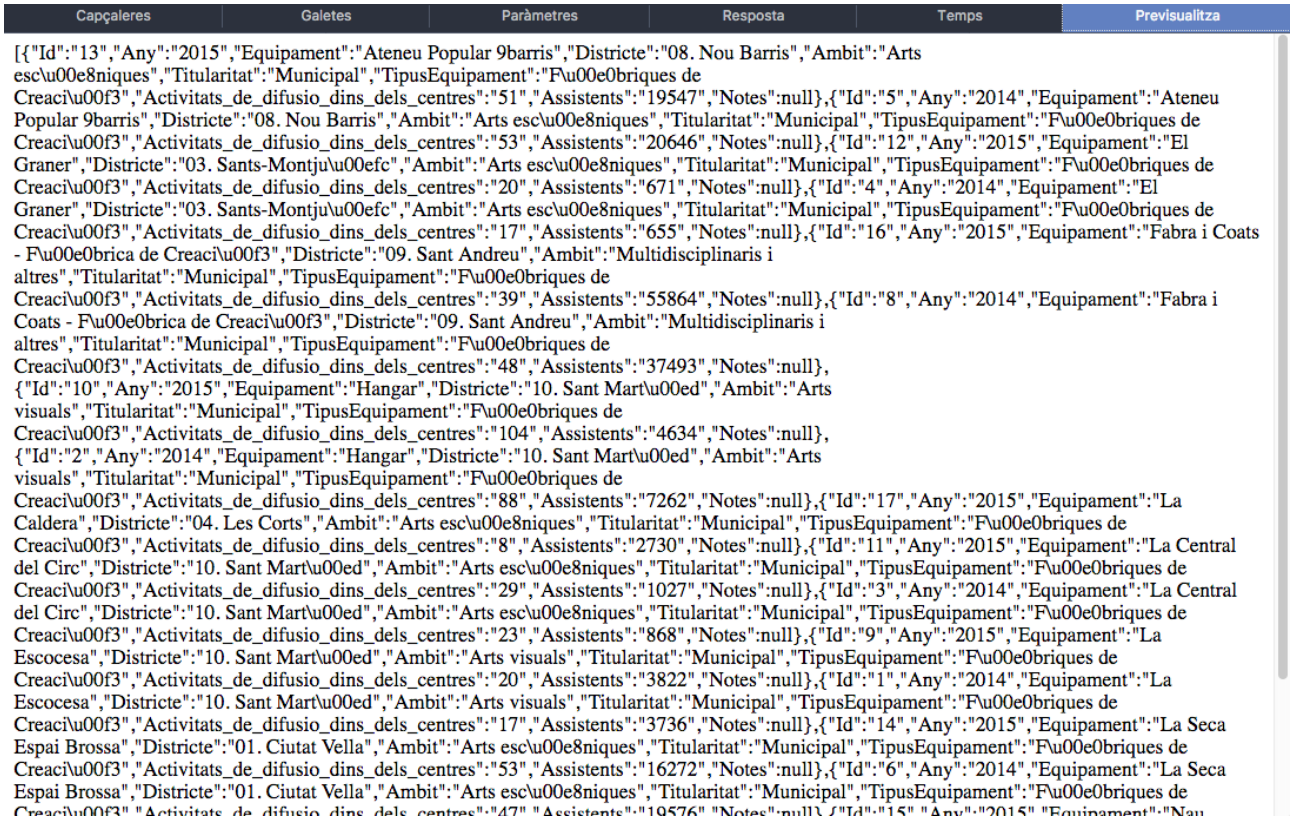


FIGURA 1.7. Previsualització d'una petició a Mozilla Firefox



Per altra banda, la pestanya "Response" ('resposta') de Google Chrome (vegeu la figura 1.8) mostra la resposta en text pla, mentre que la pestanya "Resposta" de Mozilla Firefox (vegeu la figura 1.9) mostra les dades estructurades per facilitar-ne la inspecció. És interessant recordar-ho perquè a l'hora de depurar una aplicació que utilitzi AJAX o Web Sockets, segons quin navegador utilitzeu us interessarà més consultar la informació a la pestanya de resposta o a la de previsualització.

FIGURA 1.8. Resposta d'una petició a Google Chrome

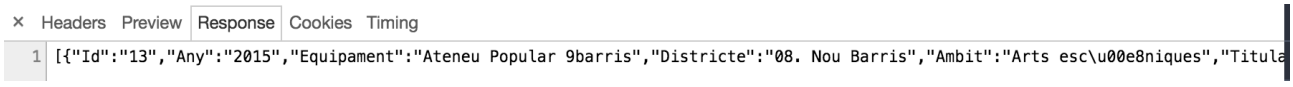
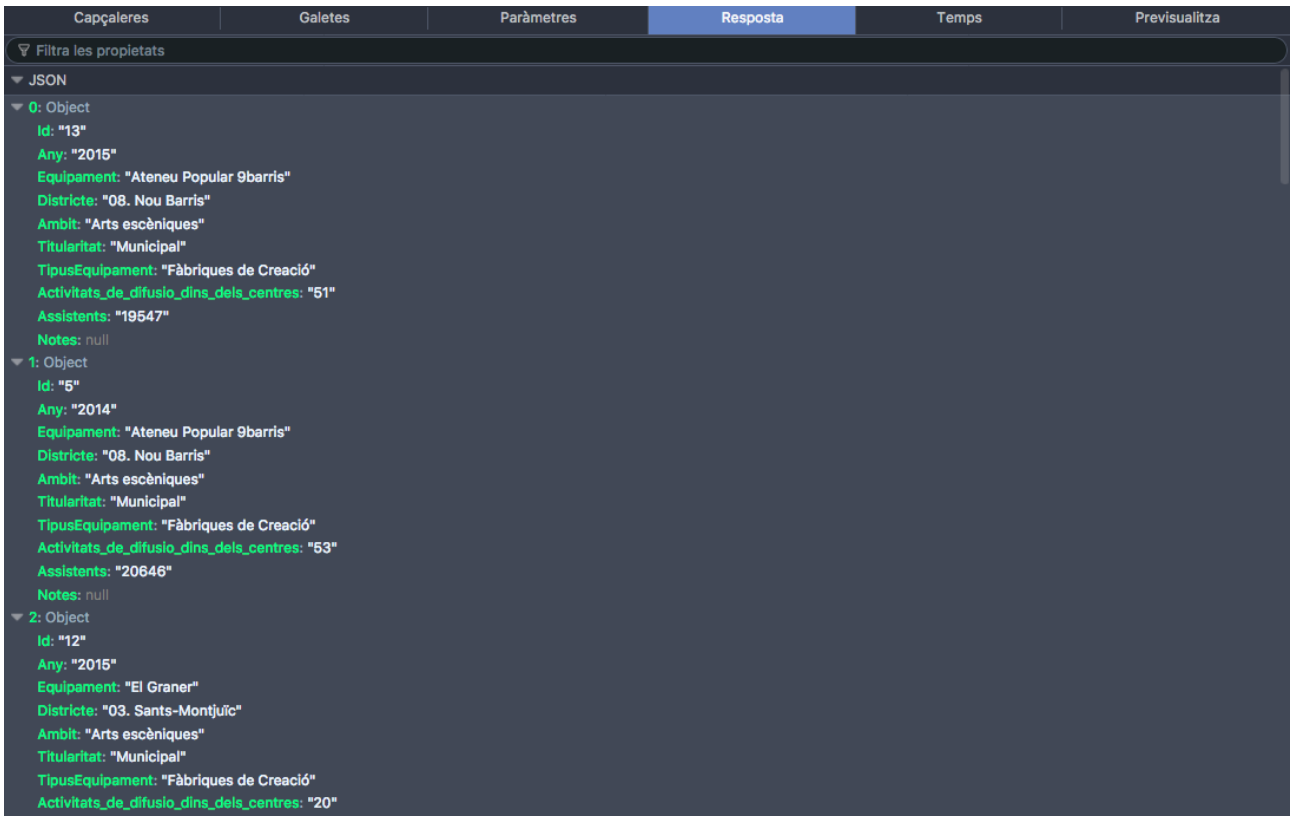


FIGURA 1.9. Resposta d'una petició a Mozilla Firefox



A la pestanya amb la informació de les galetes es poden veure les galetes de la sol·licitud d'una manera pràcticament idèntica a tots dos navegadors (vegeu la figura 1.10 i la figura 1.11). Aquesta pestanya només es mostra quan la petició inclou les galetes.

Cal destacar que es pot trobar més informació sobre les galetes a la pestanya *Application* a Google Chrome i a la pestanya *Storage* a Mozilla Firefox.

FIGURA 1.10. Galetes d'una petició a Google Chrome

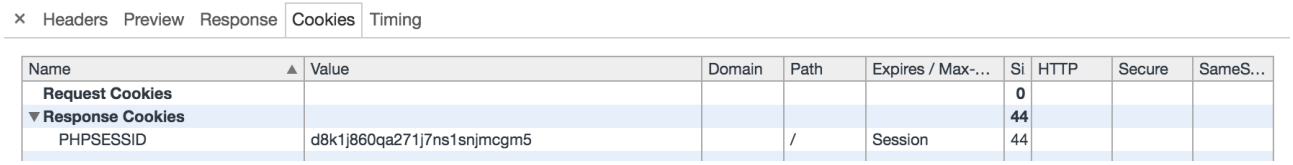
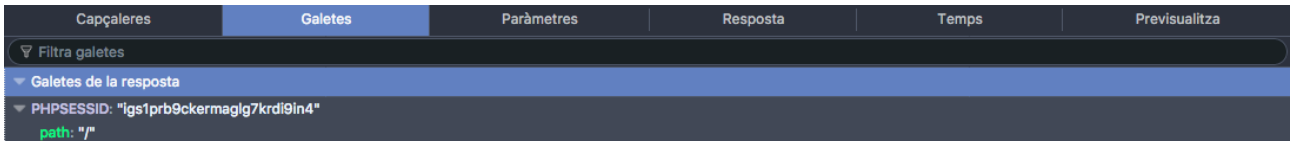


FIGURA 1.11. Galetes d'una petició a Mozilla Firefox



Finalment, es troba la pestanya *Temps*, que mostra com s'ha invertit el temps que ha trigat a realitzar-se la petició i en retornar la resposta. Aquesta pestanya mostra la mateixa informació tant a Google Chrome (vegeu la figura 1.12) com a Mozilla Firefox (vegeu la figura 1.13).

FIGURA 1.12. Temps invertit d'una petició a Google Chrome

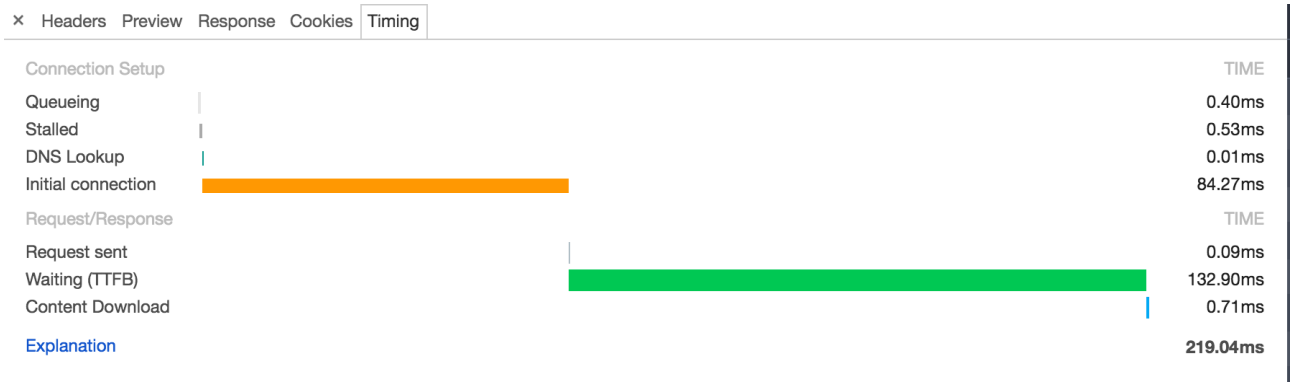
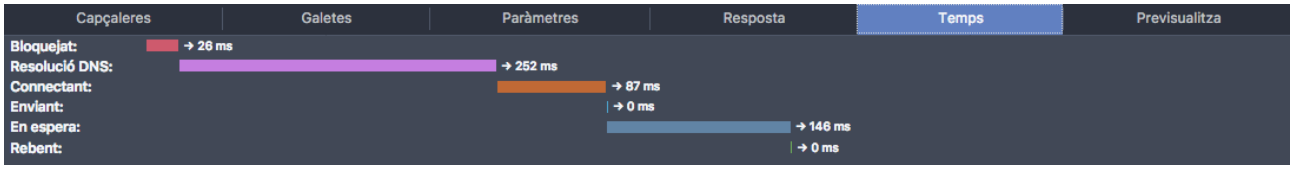


FIGURA 1.13. Temps invertit d'una petició a Mozilla Firefox



Cal destacar que Mozilla Firefox inclou una pestanya extra en la qual es mostren els paràmetres enviats, mentre que a Google Chrome es poden trobar a la mateixa pestanya de la capçalera.

Conèixer aquestes eines i les opcions que ofereixen, tant per l'enviament com per la resposta, és imprescindible per poder depurar correctament aplicacions que facin servir AJAX, ja que ens permet comprovar, entre altres coses, si la petició ha retornat un codi d'èxit o d'error, si el servidor admet CORS o no, o si els paràmetres enviats i la resposta són els esperats.

2. Programació de la comunicació asíncrona amb jQuery

A banda de la gestió d'*events* i la manipulació del DOM, una de les funcionalitats més importants de jQuery és la gestió de peticions AJAX (Javascript asíncron i XML).

Una de les diferències és que no cal fer cap comprovació prèvia per discriminar entre les versions antigues del navegador Internet Explorer i els navegadors actuals, la biblioteca s'encarrega de fer-ho, i la gestió de la resposta se simplifica mitjançant mètodes propis en lloc d'haver d'afegir-hi la detecció d'*events* (que els navegadors antics potser no admeten).

Cal destacar que per poder utilitzar aquestes funcionalitats s'ha de carregar la biblioteca jQuery. Per exemple, a partir del codi local (suposem que es troba dins del directori /js/biblioteques i que el nom del fitxer és `jquery-3.1.0.js`):

```
1 <script src="js/biblioteques/jquery-3.1.0.js"></script>
```

En cas d'utilitzar una xarxa de lliurament de continguts (en anglès, *content delivery network*, CDN) el codi seria el següent:

```
1 <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
```

S'ha de tenir en compte que tant si s'utilitza la biblioteca com si es fa servir una implementació pròpia, els navegadors apliquen la política del mateix origen i, per consegüent, només es poden realitzar peticions al mateix domini en què es troba l'aplicació, motiu pel qual la majoria dels exemples que es troben a CodePen no s'executaran correctament.

Aquesta limitació no s'aplica si el servei web al qual s'envia la petició admet CORS (peticions amb diferent origen) i, en cas de requerir-les, les capçaleres i les dades d'autenticació són correctes, o si admet JSONP (JSON amb *padding*) i es realitza la petició utilitzant aquesta tècnica.

A diferència de les implementacions pròpies, amb jQuery s'utilitza el mètode `ajax` de l'objecte `jQuery` tant per a les peticions ordinàries com per a les peticions mitjançant JSONP.

2.1 Creació i enviament de peticions amb jQuery

Com que la biblioteca jQuery s'encarrega de gestionar les diferències entre versions no cal comprovar si el navegador admet o no la utilització de l'objecte `XMLHttpRequest` o si es tracta d'una versió antiga d'Internet Explorer.

Només cal afegir la càrrega de la biblioteca (en el següent exemple mitjançant un CDN):

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
  </head>

  <body>
  </body>

</html>
```

Per evitar la complexitat de crear un servidor per servir les peticions als exemples s'utilitzaran fitxers amb format XML, que s'utilitzaran per llegir la informació. El primer d'aquests fitxers ha de dur el nom `provincies.xml` i el contingut següent:

```
<provincies>
  <provincia>Barcelona</provincia>
  <provincia>Girona</provincia>
  <provincia>Lleida</provincia>
  <provincia>Tarragona</provincia>
</provincies>
```

Una vegada tingueu l'entorn preparat (el fitxer HTML i el fitxer XML) només heu d'afegir les següents línies al final de l'element `body` per enviar la petició i mostrar el resultat per la consola:

```
<script>
  $.ajax('provincies.xml', {
    success: processarResposta,
    error: processarError
  });

  function processarResposta(dades, statusText, jqXHR) {
    console.log(dades, statusText);
  }

  function processarError(jqXHR, statusText, error) {
    console.log(error, statusText);
  }
</script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/zKVzwG?editors=1010.

Configuració de la funció 'ajax' de jQuery

Podeu consultar una llista completa d'opcions de configuració per la funció `ajax` a l'enllaç següent: www.goo.gl/fDsgT5.

Com es pot apreciar, el codi és molt simple. S'invoca el mètode `ajax` de l'objecte `jQuery` passant com a paràmetres l'URL on s'enviarà la petició i un objecte de JavaScript amb una propietat anomenada `success`, a la qual s'ha assignat la funció `processarResposta`, i una altra propietat anomenada `error`, a la qual s'ha assignat la funció `processarError`.

Aquest objecte conté les opcions que s'utilitzen per configurar la petició. En aquest cas se n'han vist dues:

- **success**: s'executa si la petició s'ha realitzat correctament i rep la resposta, el text d'estat i l'objecte `jqXHR` creat per la petició.

- **error**: s'executa quan s'ha produït un error. En aquest cas els paràmetres són l'objecte `jqXHR`, el text d'estat i l'error produït.

Objectes diferits

Podeu trobar més informació sobre els objectes diferits a l'enllaç següent: www.goo.gl/5oxmwa.

L'objecte `jqXHR` (una versió ampliada de l'objecte XMLHttpRequest dels navegadors) és retornat per la invocació del mètode `ajax` i permet realitzar o encadenar altres accions a la invocació mitjançant una interfície anomenada *objecte diferit* (*deferred object*). Per aquesta raó, es passa també com a argument les funcions `success` i `error`, de manera que es pot obtenir més informació o realitzar altres operacions sobre la petició.

Tot i que la petició es realitza automàticament en invocar el mètode `ajax`, cal recordar que (per defecte) és asíncrona i, per tant, la resposta no s'obté immediatament; l'aplicació continuarà executant-se sense bloquejar-se fins que es rebi la resposta, llavors s'executarà el mètode pertinent (assignat a `success` o `error`).

Alternativament, es pot invocar el mètode `ajax` utilitzant, només, l'objecte amb les opcions com a paràmetre i assignant l'URL a la propietat `url`:

```
$.ajax({
  url: 'provincies.xml',
  success: processarResposta,
  error: processarError
});
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/Bprxvj?editors=1010.

2.1.1 Paràmetres

Habitualment en fer una petició AJAX s'han de passar un o més paràmetres al servidor. Aquests paràmetres són visibles a l'URL en el cas d'utilitzar el mètode `GET` per a l'enviament (el mètode per defecte).

Per exemple, la següent petició s'efectuaria sobre l'URL `http://example.com` i els paràmetres enviats són `nom` i `cognom`. Com que no s'especifica el mètode, s'utilitzarà `GET`. Fixeu-vos que en aquest cas s'afegiran a l'URL: `http://example.com/?nom=Maria&Cognom=Campmany`.

```
$.ajax({
   url: 'http://example.com',
   data: { nom:"Maria", cognom:"Campmany"}
 });
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/jMgOpJ?editors=0010.

Podeu trobar més informació sobre les eines de desenvolupador a l'apartat "Comunicació asíncrona amb JavaScript" d'aquesta mateixa unitat.

Obriu les eines de desenvolupadors i a la pestanya xarxa (*network*) trobareu una línia corresponent a aquesta petició. Quan hi feu clic, tindreu accés a tota la informació sobre la petició. Pareu especial atenció a la capçalera (*header*).

El símbol `?` indica que el que hi ha a continuació són els paràmetres de la consulta, que estarà formada per parells de clau i valor separats pel signe igual. Aquests parells, al seu torn, són separats d'altres paràmetres pel símbol `&`.

Així doncs, analitzant l'URL, es pot dividir en diferents parts:

- **http**: correspon al protocol HTTP.
- **example.com**: correspon al domini al qual s'envia la petició.
- **/**: correspon a la ruta, que en aquest cas és l'arrel del domini.
- **nom=Maria**: primer paràmetre, corresponent a la clau `nom` amb valor `Maria`.
- **cognom=Campmany**: segon paràmetre, corresponent a la clau `cognom` amb valor `Campmany`.

Una alternativa a fer servir un objecte per passar els paràmetres és fer servir una cadena de text amb els parells clau-valor separats pel signe igual i aquests separats d'altres parells pel símbol `&`:

```
1 $.ajax({
2    url: 'http://example.com',
3    data: 'nom=Maria&cognom=Campmany'
4  });
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/MJVGMr?editors=0010.

Quan es fa servir un altre mètode, com `POST` o `PUT`, aquesta informació només és visible a la capçalera. Cal destacar que això no és cap garantia de privadesa ni seguretat, ja que aquesta capçalera pot ser inspeccionada per qualsevol node de la xarxa per on passi la informació.

```
1 $.ajax({
2    url: 'http://example.com',
3    method: 'post',
4    data: { nom:"Maria", cognom:"Campmany"}
5  });
```

Podeu veure aquest exemple en l'enllaç següent: www.http://codepen.io/ioc-daw-m06/pen/PGMovd?editors=0010.

En aquest cas, en analitzar la petició amb les eines de desenvolupador, podeu comprovar que l'URL no s'ha modificat i aquests paràmetres s'han afegit a la capçalera com a dades de formulari.

Cal destacar que la biblioteca jQuery permet assignar la propietat `data` de les opcions com a cadena de text per a tots els mètodes i no només per al mètode `GET`, ja que jQuery fa les conversions necessàries. Així doncs, és possible fer servir com a `data` la cadena de text `nom=Maria&cognom=Campmany` i en enviar la petició aquests paràmetres s'afegeixen com a dades a la capçalera:

```
$.ajax({
   url: 'http://example.com',
   method: 'post',
   data: 'nom=Maria&cognom=Campmany'
 });
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ORKPJa?editors=0010.

2.1.2 Tipus de dades: dataType

Si no s'especifica una altra cosa, s'espera que el format de la resposta sigui XML vàlid. És a dir, si proveu de fer una petició demanant un fitxer de text pla amb el contingut que trobareu a continuació, es produirà un error:

```
Barcelona
Girona
Lleida
Tarragona
```

Com que no s'ha especificat el tipus de dades, es produeix un error: el format del fitxer no és vàlid. Així doncs, cal especificar el tipus de dades que s'espera rebre del servidor assignant la propietat `dataType` a les opcions:

```
$.ajax({
  url: 'provincies.text',
  dataType: 'text',
  success: processarResposta,
  error: processarError
});
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/MjMvoJ?editors=1010.

Els tipus de dades admesos per les peticions AJAX de jQuery són:

- **xml** (tipus per defecte): retorna un document XML que pot ser processat per jQuery.
- **html**: retorna el codi HTML com a text pla.
- **script**: avalua la resposta com a JavaScript.
- **json**: avalua la resposta com a JSON i retorna un objecte de JavaScript.
- **jsonp**: utilitzat per obtenir les dades d'un altre domini fent servir la tècnica JSONP, retorna un objecte de JavaScript amb les dades.
- **text**: resposta com a text pla.

> En cas de carregar fitxers, com en aquests exemples, o si el servidor no especifica el tipus multimèdia de la resposta, s'ha de sobreescriure mitjançant el mètode `overrideMimeType` de l'objecte `jqXHR`.

Per assegurar que el tipus de contingut multimèdia és correcte es pot utilitzar l'opció `beforeSend`, que permet especificar una funció que serà invocada abans de realitzar la petició. Aquesta funció rep com a paràmetre l'objecte `jqXHR` i sobre aquest es pot invocar el mètode `overrideMimeType` per forçar el tipus de la resposta:

```
$.ajax({
  url: 'provincies.txt',
  dataType: 'text',
  beforeSend: function (jqXHR) {
    jqXHR.overrideMimeType('text/plain');
  },
  success: processarResposta,
  error: processarError
});
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ozremJ?editors=1010.

Cal destacar que el més habitual és que el servidor web retorni el tipus multimèdia correcte i no s'hagi de sobreescriure.

Com es pot apreciar, un altre dels tipus admesos és HTML.

Suposem que s'ha d'afegir dinàmicament a una pàgina el codi HTML següent (podria ser la resposta enviada pel servidor, per exemple):

```
<html>
  <h1>Provincies</h1>
  <ul>
    <li>Barcelona</li>
    <li>Girona</li>
    <li>Lleida</li>
    <li>Tarragona</li>
  </ul>
</html>
```

Per fer-ho, només caldria especificar el tipus com `html` i afegir-lo utilitzant el mètode `append` de l'objecte jQuery:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.ajax({
      url: 'provincies.html',
      dataType: 'html',
      success: processarResposta,
      error: processarError
```

```
    });

    function processarResposta(dades, statusText, jqXHR) {
        $dades = $(dades);
        $('body').append($dades);

    }

    function processarError(jqXHR, statusText, error) {
      console.log(error, statusText);
    }
  </script>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/XjLaVe?editors=1010.

Errors habituals quan no s'especifica el tipus de dades

Habitualment es produeixen errors si no s'especifica el tipus de dades mitjançant la propietat `dataType`, ja que a l'hora de processar la resposta la biblioteca jQuery intentarà interpretar-lo com XML i si no ho aconsegueix llença una excepció.

En aquest cas, com que el codi HTML utilitzat té el mateix format que XML (que és el valor per defecte de dataType) es considera correcte i no cal sobreescriure el tipus multimèdia.

En canvi, si la resposta fos en format JSON, caldria canviar el `dataType` per JSON i sobreescriure el tipus multimèdia. A diferència de les implementacions manuals, no cal analitzar la resposta perquè ja és retornada com un objecte de JavaScript. Proveu de crear un fitxer anomenat `provincies.json` amb el contingut següent:

```
{
  "provincies": [
    {"nom": "Barcelona", "cp": "08"},
    {"nom": "Girona", "cp": "17"},
    {"nom": "Lleida", "cp": "25"},
    {"nom": "Tarragona", "cp": "43"}
  ]
}
```

I substituir el codi JavaScript pel següent:

```
$.ajax({
  url: 'provincies.json',
  dataType: 'json',
  beforeSend: function (jqXHR) {
    jqXHR.overrideMimeType('application/json');
  },
  success: processarResposta,
  error: processarError
});

function processarResposta(dades, statusText, jqXHR) {
  console.log(dades, statusText);
}

function processarError(jqXHR, statusText, error) {
  console.log(error, statusText);
}
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wzLqOM?editors=1010.

Com es pot apreciar, en aquest cas la resposta és un objecte JavaScript que es pot processar de la forma habitual.

2.1.3 Mètodes d'enviament

De la mateixa manera que a les implementacions pròpies d'AJAX, la biblioteca jQuery permet especificar el mètode amb el qual s'envien les peticions. Només cal afegir, a l'objecte amb les opcions, la propietat `method` i assignar-li el mètode pertinent: `GET` (per defecte), `POST`, `PUT` o `DELETE`. Per exemple, es pot especificar el mètode `GET` per llegir un fitxer JSON:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.ajax({
      method: 'get',
      url: 'provincies.json',
      dataType: 'json',
      beforeSend: function (jqXHR) {
        jqXHR.overrideMimeType('application/json');
      },
      success: processarResposta,
      error: processarError
    });

    function processarResposta(dades, statusText, jqXHR) {
      console.log(dades, statusText);
    }

    function processarError(jqXHR, statusText, error) {
      console.log(error, statusText);
    }
  </script>
</body>

</html>
```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-daw-m06/pen/wzLQOR?editors=1010.

Cal recordar que aquests mètodes es corresponen amb els del protocol HTTP 1.1, i als serveis web REST són utilitzats sobre un mateix URL per realitzar diferents accions:

- **GET**: consultar les dades.
- **POST**: afegir noves dades.

- **PATCH**: actualitzar dades ja existents.
- **PUT**: reemplaçar dades ja existents.
- **DELETE**: eliminar dades.

'get' i 'post'

Podeu trobar totes les opcions dels mètodes `$.get` i `$.post` als següents enllaços: www.goo.gl/DVzxg8 i www.goo.gl/zRbp5A; respectivament.

A més a més, jQuery ofereix dues dreceres per simplificar l'enviament de peticions a través dels mètodes `GET` i `POST`. Tots dos mètodes s'invoquen a partir de l'objecte jQuery:

- `$.get`: envia una petició AJAX fent servir el mètode `GET`.
- `$.post`: envia una petició AJAX fent servir el mètode `POST`.

Aquests mètodes ofereixen un nombre més limitat d'opcions, però suficient en la major part de les situacions:

- **url**: una cadena amb l'URL al qual s'envia la petició.
- **data**: un objecte o cadena de text que s'enviaran com a paràmetres de la petició.
- **sucess**: funció de tipus *callback* que serà cridada si la petició es realitza amb èxit.
- **dataType**: el tipus de dades esperat pel servidor (`xml`, `json`, `script`, `text` i `html`).

Aquestes opcions es poden passar com a paràmetres individuals:

```
$.get(url, data, success, dataType);
```

O com un objecte en el qual totes les propietats, excepte `url`, són opcionals:

```
$.get({
  url: 'provincies.html',
  data: {nom: 'Josep', cognoms: 'Torres Blanc'},
  success: processarResposta},
  dataType: 'html'
});
```

Així doncs, per fer una petició amb el mètode `GET` i obtenir una resposta en HTML el codi complet seria el següent:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.get({
      url: 'provincies.html',
```

```
      data: {nom: 'Josep', cognoms: 'Torres Blanc'},
      success: processarResposta,
      dataType: 'html'
    });

    function processarResposta(dades, statusText, jqXHR) {
      $dades = $(dades);
      $('body').append($dades);
    }
  </script>
</body>

</html>
```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-daw-m06/pen/RGzvgP?editors=1010.

Com es pot apreciar, en aquest cas no s'utilitzen els paràmetres enviats, ja que no hi ha cap servidor servint la resposta, però a les eines de desenvolupador podeu comprovar que la capçalera ha inclòs aquests paràmetres. En tractar-se del mètode `get` s'ha convertit l'URL de la petició en quelcom similar al següent: `/provincies.html?nom=Josep&cognoms=Torres%20Blanc`.

És a dir, jQuery ha convertit l'objecte passat com a argument en paràmetres vàlids per enviar com a petició, fent servir el nom de les propietats com a clau i el valor assignat com a valor del paràmetre.

S'ha de tenir en compte que, tal com passa amb el mètode `ajax`, si l'URL de la petició no correspon a un servidor que envia una resposta amb un tipus especificat, l'aplicació esperarà que es tracti d'una resposta XML i donarà error. En el cas de les dreceres no existeix l'opció `beforeSend` i, per consegüent, no es pot sobreescriure el tipus multimèdia de la resposta.

2.1.4 Interpretar la resposta

Segons el tipus de data (`dataType`) de la resposta i si aquesta és correcta, la resposta serà retornada de diferents maneres:

- **XML**: un objecte `XMLDocument`.
- **HTML**, **TEXT**: una cadena de text amb el contingut.
- **JSON**: un objecte de JavaScript.
- **Script**: un bloc de codi JavaScript que serà avaluat (executat).

`XMLDocument` és un tipus d'objecte que és herència de la interfície del DOM `document` i no afegeixen cap altra propietat o mètode nous. Això permet utilitzar jQuery per manipular aquest document:

```
<!DOCTYPE html>
<html>
```

```
3
4  <head>
5    <meta charset="utf-8">
6    <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
7  </head>
8
9  <body>
10   <script>
11     $.get({
12       url: 'provincies.xml',
13       success: processarResposta,
14     });
15
16     function processarResposta(dadesXML, statusText, jqXHR) {
17         var $provincies = $(dadesXML).find('provincia');
18         var $llista = $('<ul>');
19
20
21         $provincies.each(function() {
22             var nomProvincia = $(this).html();
23             $provincia = $('<li>');
24             $provincia.html(nomProvincia);
25             $llista.append($provincia);
26         })
27
28         $('body').append($llista);
29     }
30   </script>
31 </body>
32
33 </html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/MjMLRP?editors=1010.

Una vegada s'ha obtingut la resposta, es converteix en un objecte jQuery i se seleccionen tots els elements amb el nom `provincia`. A continuació, es crea una nova llista sense numeració (`<ul>`) i es procedeix a recórrer-los fent servir el mètode `each`, que invoca la funció passada com a argument amb l'element corresponent com a context. És a dir, l'element és referenciat per `this` (en aquest cas, cadascun dels elements de `provincies`).

De cadascun d'aquests elements s'obté el nom de la província a partir del contingut intern de l'element mitjançant el mètode `html`. Seguidament es crea un nou element de tipus element de llista (`<li>`) i s'assigna com a contingut d'aquest element el nom de la província que, finalment, és afegit a la llista que, al seu torn, és afegida al cos de la pàgina quan es finalitza el recorregut.

Com que XML es el tipus per defecte, no cal especificar-lo quan es treballa amb aquest format.

En el cas de `HTML` es pot utilitzar la funció `jQuery` per convertir aquest text en un objecte jQuery que, al seu torn, converteix la cadena de text en elements del DOM: `$branca = $(dades)`. Aquesta nova branca es pot afegir a altres elements: `$('body').append($branca)`.

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5    <meta charset="utf-8">
```

```
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.ajax({
      url: 'provincies.html',
      beforeSend: function (jqXHR) {
        jqXHR.overrideMimeType('text/html');
      },
      success: processarResposta,
      dataType: 'html'
    });

    function processarResposta(dadesHTML, statusText, jqXHR) {
        var $branca = $(dadesHTML);
        $('body').append($branca);
    }
  </script>
</body>

</html>
```

Podeu veure aquest exemple en lenllaç següent: www.codepen.io/ioc-daw-m06/pen/EgBMXj?editors=1010.

Com es pot apreciar, s'ha fet servir el mètode `ajax` per incloure l'opció `beforeSend` i canviar el tipus multimèdia a `text/html`. D'aquesta manera s'eviten possibles errors de format, ja que el sistema de fitxers no especifica el tipus de contingut.

Per analitzar un fitxer de tipus textual s'han de fer servir les funcions de tractament de cadenes, com per exemple `split`, o expressions regulars. Per exemple, si el contingut de la resposta fos la següent:

```
Barcelona,Girona,Lleida,Tarragona
```

Es podria processar la resposta dividint la cadena per la coma, mitjançant el mètode `split`:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.ajax({
      url: 'provincies.txt',
      beforeSend: function(jqXHR) {
        jqXHR.overrideMimeType('text/plain');
      },
      success: processarResposta,
      dataType: 'text'
    });

    function processarResposta(dadesText, statusText, jqXHR) {
      var provincies = dadesText.split(',');
      var $llista = $('<ul>');

      for (var i = 0; i < provincies.length; i++) {
```

```
        var $item = $('<li>');
        $item.html(provincies[i]);
        $llista.append($item);
      }

      $('body').append($llista);
    }
  </script>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/ZpdPME?editors=1010.

En el cas que la resposta sigui un objecte JavaScript és molt fàcil treballar-hi, ja que es pot accedir directament a tota la informació fent servir la notació de claudàtors (`provincia['nom']`) o de punt (`provincia.nom`).

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
  <script>
    $.ajax({
      url: 'provincies.json',
      beforeSend: function(jqXHR) {
        jqXHR.overrideMimeType('application/json');
      },
      success: processarResposta,
      dataType: 'json'
    });

    function processarResposta(dadesJSON, statusText, jqXHR) {
      var provincies = dadesJSON.provincies;
      var $llista = $('<ul>');

      for (var i = 0; i < provincies.length; i++) {
        var $item = $('<li>');
        $item.html(provincies[i].nom + " (" + provincies[i].cp + ")");
        $llista.append($item);
      }

      $('body').append($llista);
    }

  </script>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/PGrLVd?editors=1010.

Per acabar, si la resposta és de tipus `script` s'ha d'anar amb molt de compte, perquè s'executarà. Afegiu un fitxer anomenat `provincies.js` amb el codi següent:

```
var provincies = ['Barcelona', 'Girona', 'Lleida', 'Tarragona'];
```

```
2
3  for (var i=0; i<provincies.length; i++) {
4    alert(provincies[i]);
5  }
```

I comproveu l'efecte amb el codi següent, que realitzarà una petició de tipus `script` i, per consegüent, la resposta serà carregada i executada immediatament:

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5    <meta charset="utf-8">
6    <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
7  </head>
8
9  <body>
10   <script>
11     $.ajax({
12       url: 'provincies.js',
13       beforeSend: function(jqXHR) {
14         jqXHR.overrideMimeType('application/javascript');
15       },
16       success: processarResposta,
17       dataType: 'script'
18     });
19
20     function processarResposta(dadesJS, statusText, jqXHR) {
21       console.log("Dades rebudes:", dadesJS);
22     }
23
24   </script>
25 </body>
26
27 </html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/pRLKZO?editors=1010.

Com es pot apreciar, l'script s'executa automàticament i, per tant, no es pot controlar els efectes que tindrà aquest tipus de resposta, ja que depèn completament del servidor. Per aquest motiu s'ha d'estar molt segur que aquest comportament és el desitjat perquè permet executar el possible codi malintencionat rebut com a resposta del servidor.

2.1.5 Events AJAX

Durant l'enviament d'una petició AJAX mitjançant la biblioteca jQuery es disparen tot un seguit d'*events*. Aquests *events* poden ser locals (propis d'una petició concreta) o globals (disparats al document).

L'ordre en què es disparen aquests *events* és el següent:

- **ajaxStart** (global): es dispara quan comença la petició i no hi ha cap altra petició en execució.

- **beforeSend** (local): es dispara abans de realitzar-se l'enviament i permet modificar la petició abans d'enviar-la (per exemple, per sobreescriure el tipus multimèdia).
- **ajaxSend** (global): disparat globalment quan es realitza l'enviament.
- **success** (local): es dispara només si la petició retorna amb una resposta exitosa.
- **ajaxSuccess** (global): disparat en les mateixes condicions que l'anterior, però globalment.
- **error** (local): disparat només quan es produeix un error.
- **ajaxError** (global): igual que l'anterior però disparat globalment.
- **complete** (local): es dispara en finalitzar la petició independentment de si ha estat exitosa o errònia.
- **ajaxComplete** (global): igual que l'anterior però disparat globalment.
- **ajaxStop** (global): es dispara quan no hi ha més peticions AJAX processant-se.

Fixeu-vos que els *events* locals es corresponen amb les opcions a les quals es pot afegir una funció de tipus *callback*: `beforeSend`, `success`, `error` i `complete`.

Per altra banda, a excepció de l'*event* `ajaxStart` i `ajaxStop`, la resta d'*events* globals són rèpliques dels *events* locals i es disparen a continuació d'aquests.

Cal destacar que per detectar quan es dispara un *event* global s'ha d'especificar la detecció al document, tal com es mostra en l'exemple següent:

```
$(document).ajaxSend(function() {
  console.log("S'ha enviat una petició AJAX");
```

Per contra, la detecció d'*event* local s'especifica a la petició concreta que hi està lligada, com es pot apreciar en l'exemple següent:

```
$.ajax({
      url: 'provincies.xml',
      beforeSend: function(jqXHR) {
        console.log("Disparat abans d'enviar una petició concreta");
      });
```

Fixeu-vos que en el primer cas el missatge es mostraria a la consola quan s'enviés qualsevol petició AJAX, mentre que en el segon cas només es mostraria quan s'enviés la petició concreta on s'ha especificat.

2.1.6 Altres opcions del mètode Ajax

El mètode `ajax` ofereix múltiples opcions per crear una petició, a banda dels més utilitzats com `method`, `url` i `data`, podeu trobar els següents, que permeten crear

peticions més avançades afegint dades d'autenticació o modificant les capçaleres per adequar-les als requeriments dels serveis web.

- **async**: per defecte el seu valor és `true`; si s'estableix com a `false`, la petició serà síncrona.
- **contentType**: permet canviar el tipus de contingut enviat en les peticions a altres dominis; si és diferent d'`application/x-www-form-urlencoded`, `multipart/form-data` o `text/plain` s'activaran mesures de comprovació extres al navegador.
- **crossDomain**: permet forçar l'enviament de la petició com si es tractés d'un altre domini, tot i que el destí sigui el mateix en què es troba l'aplicació.
- **headers**: aquesta propietat permet afegir parelles de claus i valors addicionals a la capçalera de la petició. Per exemple, és possible que alguns serveis web requereixin que a la capçalera es trobi un *token* d'autenticació.
- **password**: contrasenya que serà utilitzada en una petició HTTP d'autenticació.
- **username**: nom d'usuari que serà utilitzat en una petició HTTP d'autenticació.

Addicionalment, **tot i que no es recomana fer-lo servir**, jQuery ofereix el mètode `ajaxSetup`, que permet establir uns valors per defecte que seran utilitzats per totes les peticions realitzades pel mètode `ajax` i els seus derivats, com els mètodes `get` i `post`.

Així, per exemple, el següent codi canviaria el mètode d'enviament per defecte (`GET`) per `POST`, cosa que podria fer que altres peticions de l'aplicació deixessin de funcionar o tinguessin efectes no desitjats.

```
1 $.ajaxSetup({
2   method: 'post'
3 });
```

2.2 JSONP i CORS amb jQuery

Sovint és necessari l'accés a dades en serveis webs amb origen diferent. Per poder accedir a aquestes dades depenem del servidor, que ha d'oferir les dades o bé en format JSONP (JSON amb *padding*) o implementar mecàniques per habilitar el CORS (Cross-Origin Resource Sharing).

En tots dos casos jQuery utilitza la mateixa interfície, mitjançant el mètode AJAX, de manera que les tècniques emprades són pràcticament transparents a l'usuari.

API de Flickr

Podeu trobar tota la informació per desenvolupadors de l'API de Flickr a l'enllaç següent: www.goo.gl/tAVTBD.

Aquesta estandardització es veu molt clarament en la implementació de les peticions amb JSONP, ja que en lloc d'afegir una etiqueta `script` amb l'origen de

les dades i passant els paràmetres com una cadena de text, es pot crear una petició AJAX amb les opcions corresponents. Fixeu-vos en una aplicació que consulta l'agregador (*feed*) de l'aplicació `flickr` enviant com a paràmetres l'etiqueta `kittens` i el format `json`.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>
  <script>
    $.ajax({
      url: 'http://api.flickr.com/services/feeds/photos_public.gne',
      success: processarResposta,
      dataType: "jsonp",

      // Nom del paràmetre que indica al servidor el nom de la funció de
          callback, definit a la seva documentació
      jsonp: "jsoncallback",

      // Paràmetres que es passen al servidor, definits a la seva documentació
      data: {
        tags: 'kitten',
        format: 'json'
      }
    });

    function processarResposta(dades) {
      var imatges = dades.items;

      for (var i = 0; i < imatges.length; i++) {
        var $img = $('<img>');
        $img.attr('src', imatges[i].media.m);
        $img.attr('title', imatges[i].title);
        $img.attr('alt', imatges[i].title);
        $('body').append($img);
      }
    };
  </script>
</body>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/PGrLVd?editors=1010.

Com es pot apreciar, només es diferencia d'una crida AJAX estàndard en el fet que s'ha especificat l'opció `jsonp` per indicar el nom del paràmetre corresponent a la funció *callback*, que es farà servir com a *padding* a la resposta, i l'opció `dataType` a la qual s'ha assignat el valor `jsonp`.

Quant a les crides a serveis web que admetin CORS, com també en les implementacions pròpies d'AJAX, no canvia res quan el servidor inclou a la capçalera el paràmetre `Access-Control-Allow-Origin` i el seu valor es correspon amb el del domini on s'executa l'aplicació o el seu valor és `*` i, per consegüent, accessible per a tothom.

Vegeu en l'exemple següent com es consulta una font de dades obertes de l'Ajuntament de Barcelona per recuperar la llista de festivals de l'any 2015 en format JSON:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
</head>

<body>
<h1>Festivals 2015</h1>

  <script>

    $.ajax({
      url: 'http://dades.eicub.net/api/1/festivals-assistents',
      beforeSend: function(jqXHR) {
        jqXHR.overrideMimeType('application/json');
      },
      success: processarResposta,
      dataType: 'json',
      data: {
        format : 'json.xml',
        Any : 2015,
      }
    });

    function processarResposta(dades, statusText, jqXHR) {
        var $llista = $('<ul>');

        for (var i=0; i<dades.length; i++) {
            var $dada = processarDada(dades[i]);
            $llista.append($dada)
        }

        $('body').append($llista);
    }

    function processarDada(dada) {
      var $item = $('<li>');
      var $enllac = $('<a>');
      $enllac.html(dada.NomDelFestival);
      $enllac.attr('href', dada.Web);
      $enllac.attr('title', dada.Organitzador);
      $item.append($enllac);

      return $item;
    }

  </script>
</body>
</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/RGXbBr?editors=1010.

En cas que el servei web requereixi autenticació o altres paràmetres, a les opcions que es passen al mètode `ajax` es pot especificar la propietat `xhrFields` per incloure-les en la petició (per exemple, per afegir l'ús de credencials si el servidor les demana), i es faria de la manera següent:

```
$.ajax({
  url: url_en_un_domini_diferent,
  xhrFields: {
    withCredentials: true
  }
});
```

Com que les dades addicionals que pot requerir un servei web que implementi les mecàniques per habilitar CORS poden ser molt diferents, cal consultar-ne la documentació per poder configurar correctament les peticions.

2.3 Accés a dades obertes amb jQuery

L'accés a dades obertes com les que ofereix l'Observatori de dades culturals de Barcelona (www.barcelonadadescultura.bcn.cat/dades-obertes) es realitza de la mateixa manera que altres peticions AJAX mitjançant jQuery, amb l'avantatge que en cas que sigui necessari enviar paràmetres, la tasca se simplifica, ja que no cal modificar la informació de la capçalera i es tracta igual si el mètode d'enviament és `GET` o qualsevol altre.

Vegeu a continuació un exemple d'una consulta que carrega les dades de les "activitats de difusió de les fàbriques de creació" i permet seleccionar, segons el seu identificador, de quina activitat es volen visualitzar les dades.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <script src="https://code.jquery.com/jquery-3.1.0.js"></script>

  <style>
    h1 {
      text-align:center;
    }

    fieldset {
      width: 300px;
      margin: 0 auto;
    }
    label {
      display: inline-block;
      width: 150px;
      font-weight: bold;
    }
    span {
      display: inline-block;
      width: 150px;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
  </style>

</head>

<body>
  <h1>Activitats de difusió de les fàbriques de creació</h1>
  <fieldset>
    <select id="identificador">
    <option>Selecciona un identificador</option></select>
    <ul>
      <li><label>Id:</label><span id="id"></span></li>
      <li><label>Any:</label><span id="any"></span></li>
      <li><label>Equipament:</label><span id="equipament"></span></li>
      <li><label>Districte:</label><span id="districte"></span></li>
```

```
      <li><label>Ambit:</label><span id="ambit"></span></li>
      <li><label>Assistents:</label><span id="asistents"></span></li>
      <li><label>Notes:</label><span id="notes"></span></li>
      <li><label>Tipus d'equipament:</label><span id="tipusEquipament"></span><
          /li>

      <li><label>Titularitat:</label><span id="titularitat"></span></li>
      <li><label>Activitats de difusió dins dels centres:</label><span id="
          activitats"></span></li>
    </ul>
  </fieldset>

  <script>
     var dades = {};

     $.ajax({
       url: 'http://dades.eicub.net/api/1/fabriquescreacio_difusio',
       beforeSend: function(jqXHR) {
         jqXHR.overrideMimeType('application/json');
       },
       success: processarResposta,
       dataType: 'json',
     });

     function processarResposta(resposta, statusText, jqXHR) {
       var $llistaDesplegable = $('#identificador');

       for (var i = 0; i < resposta.length; i++) {
         var $item = processarDada(resposta[i]);
         dades[resposta[i].Id] = resposta[i];
         $llistaDesplegable.append($item);
       }

       $llistaDesplegable.on('change', actualitzarDadesMostrades);
     }

     function processarDada(dada) {
       var $item = $('<option>');
       $item.attr('value', dada.Id);
       $item.html(dada.Id);
       return $item;
     }

     function actualitzarDadesMostrades(event) {
       var $llistaDesplegable = $('#identificador');
       var dada = dades[$llistaDesplegable.val()]

       actualitzarDadaMostrada('id', dada.Id);
       actualitzarDadaMostrada('any', dada.Any);
       actualitzarDadaMostrada('equipament', dada.Equipament);
       actualitzarDadaMostrada('districte', dada.Districte);
       actualitzarDadaMostrada('ambit', dada.Ambit);
       actualitzarDadaMostrada('asistents', dada.Assistents);
       actualitzarDadaMostrada('notes', dada.Notes || 'No hi ha cap nota');
       actualitzarDadaMostrada('tipusEquipament', dada.TipusEquipament);
       actualitzarDadaMostrada('titularitat', dada.Titularitat);
       actualitzarDadaMostrada('activitats', dada.
           Activitats_de_difusio_dins_dels_centres);

     }

     function actualitzarDadaMostrada(nom, valor) {
       $('#' + nom).html(valor);
     }

  </script>
</body>

</html>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/bwXGLx.

Com es pot apreciar, en aquest cas no s'han especificat paràmetres perquè la font de dades no els requeria. En els casos que és necessari només cal especificar a l'objecte d'opcions la propietat `data` i assignar els paràmetres necessaris, o en forma de cadena de text o d'objecte, i la biblioteca s'encarregarà d'afegir-los a la capçalera de la forma correcta segons el mètode emprat.

2.4 Mètodes d'ajuda: serialize, serializeArray i param

La biblioteca jQuery ofereix un seguit de mètodes per facilitar l'obtenció i manipulació de dades a l'hora de realitzar peticions, entre els quals cal destacar `serialize`, `serializeArray` i `params`.

El mètode `serialize` converteix la informació dels controls d'un objecte `jQuery` que encapsula un formulari en una cadena de text codificada per fer servir com a paràmetre. Aquesta cadena estarà formada pels controls que disposin de la propietat `name` i el seu valor.

```
<form>
  <input name="nom" />
  <input name="cognom" />
</form>

<div></div>

<script>
  $('input').on('input change', function() {
    var serialitzat = $('form').serialize();
    $('div').html(serialitzat);
  });
</script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/vXoExz?editors=1010.

Com es pot apreciar, això permet establir l'opció `data` d'una petició AJAX directament a partir d'aquesta cadena, ja que inclou tota la informació del formulari sense haver de recórrer els elements i extreure'n les dades una per una.

De forma similar, el mètode `serializeArray` també obté la informació del formulari, però en aquest cas genera un *array* d'objectes JavaScript amb tota la informació del formulari.

```
<form>
  <input name="nom" value="Maria"/>
  <input name="cognom" value="Campmany"/>
</form>
<button>Comprovar</button>

<script>
  $('button').on('click', function() {
    var serialitzat = $('form').serializeArray();
```

```
    console.log(serialitzat);
  });
</script>
```

Podeu veure aquest exemple en l'enllaç següent: www.codepen.io/ioc-daw-m06/pen/wgbobe?editors=1011.

L'*array* d'objectes generats per la invocació del mètode `serializeArray` seria el següent:

```
[
  Object {
    name: "nom",
    value: "Maria"
  },
  Object {
    name: "cognom",
    value: "Campmany"
  }
]
```

Fixeu-vos que aquest objecte es pot utilitzar directament com a valor assignat a l'opció `data`: aquesta opció accepta tant objectes de JavaScript (sense mètodes) com cadenes de text.

Cadena de consulta és una mena de cadena de text amb un format especial utilitzat per enviar peticions i que, inclús, pot portar paràmetres.

Mentre que `serialize` i `serializeArray` faciliten la tasca de convertir les dades introduïdes als controls d'un formulari per fer-les servir com a **dades** en una petició AJAX, el mètode `param` permet convertir objectes de JavaScript, objectes de jQuery i *arrays* en una **cadena de consulta** codificada:

```
var dades = [{
    name: 'nom',
    value: 'Maria'
  }, {
    name: 'cognom',
    value: 'Campmany'
  }, ]

  console.log($.param(dades));
```

Podeu veure aquest exemple en el següent enllaç: www.codepen.io/ioc-daw-m06/pen/xEvbpp?editors=0012.

La cadena de consulta obtinguda en invocar el mètode `param` serà: `nom=Maria&cognom=Campmany`.

Fixeu-vos que mentre que `serialize` i `serializeArray` són mètodes que es criden sobre una instància d'objecte `jQuery` que conté la referència al formulari, el mètode `param` s'invoca directament a partir de la funció `jQuery` passant les dades que s'han de convertir com a argument.

S'ha de tenir en compte que en el cas dels objectes JavaScript, han de tenir forçosament el format següent:

```
[
  {
    name: nom_del_parametre
    value: valor_corresponent
```

```
  }
]
```

Els objectes jQuery han de contenir referències a objectes de tipus `input` en què els controls tinguin definit l'atribut `name` (com en el cas de `serialize` i `serializeArray`), en cas contrari no funcionarà correctament.