Asynchronous JavaScript And XML (AJAX)

•Ajax es una técnica de desarrollo web que permite enviar consultas y recibir respuestas del servidor sin tener que volver a cargar toda la web en el navegador.

•Para enviar y recibir las consultas se utiliza el objeto JavaScript XMLHttpRequest o el método Fetch.

•Originalmente las respuestas del servidor se enviaban codificadas en formato XML. Actualmente se utiliza para codificación JSON, XML, HTML y archivos de texto.

•Mediante JavaScript deberemos extraer los datos devueltos por el servidor y mostrárselos al usuario (o realizar las acciones pertinentes).

AJAX

Métodos Nativos

•ActiveXObject (IE8 e inferiores, esta deprecado)

•XMLHttpRequest

•API Fetch

Librerías Externas

•jQuery.ajax()

•Axios

•etc.

AJAX no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen:

•HTML y CSS, para crear una presentación basada en estándares.

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

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

•XMLHttpRequest o Fetch, para el intercambio asíncrono de información.

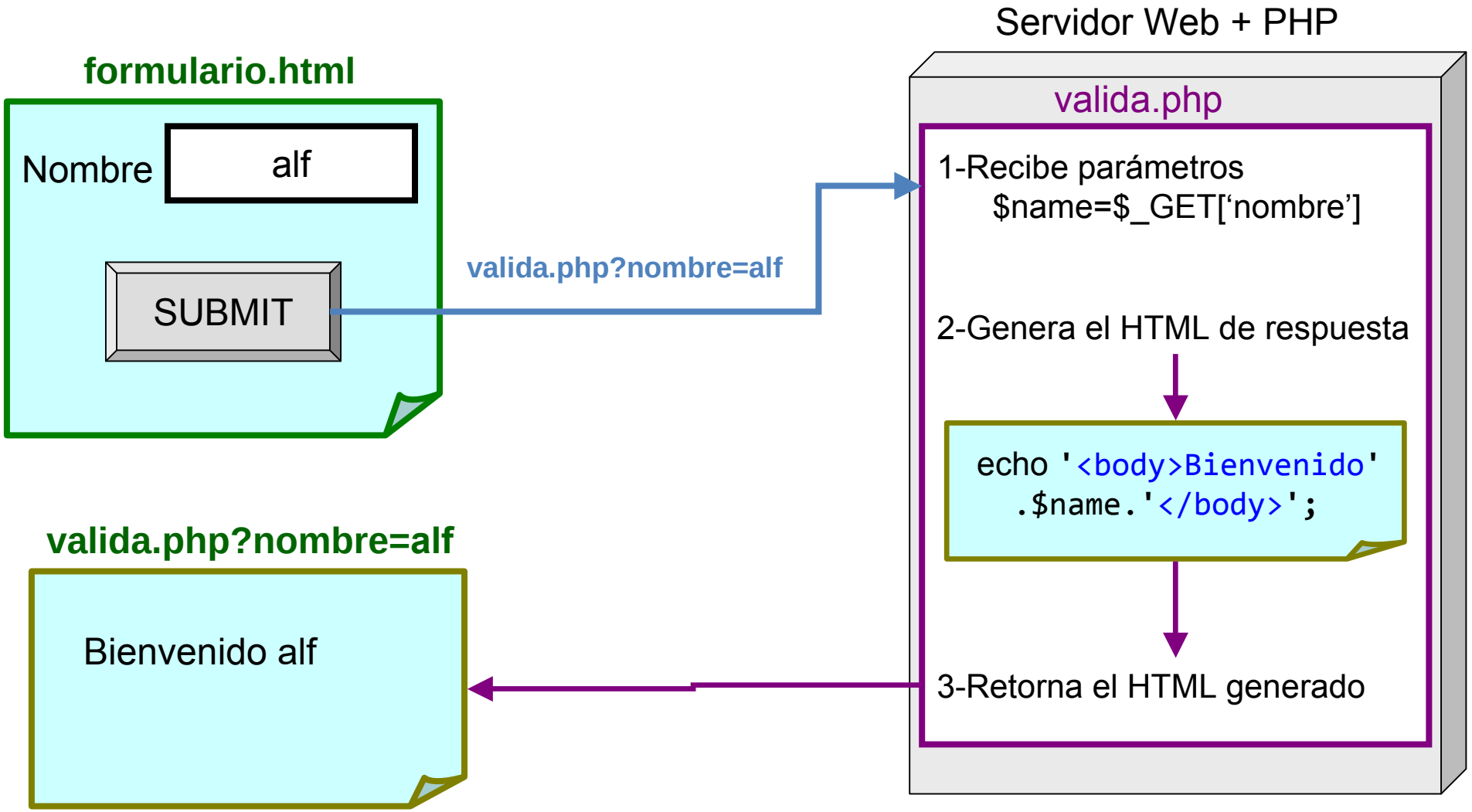
•JavaScript, para unir todas las demás tecnologías.

AJAX

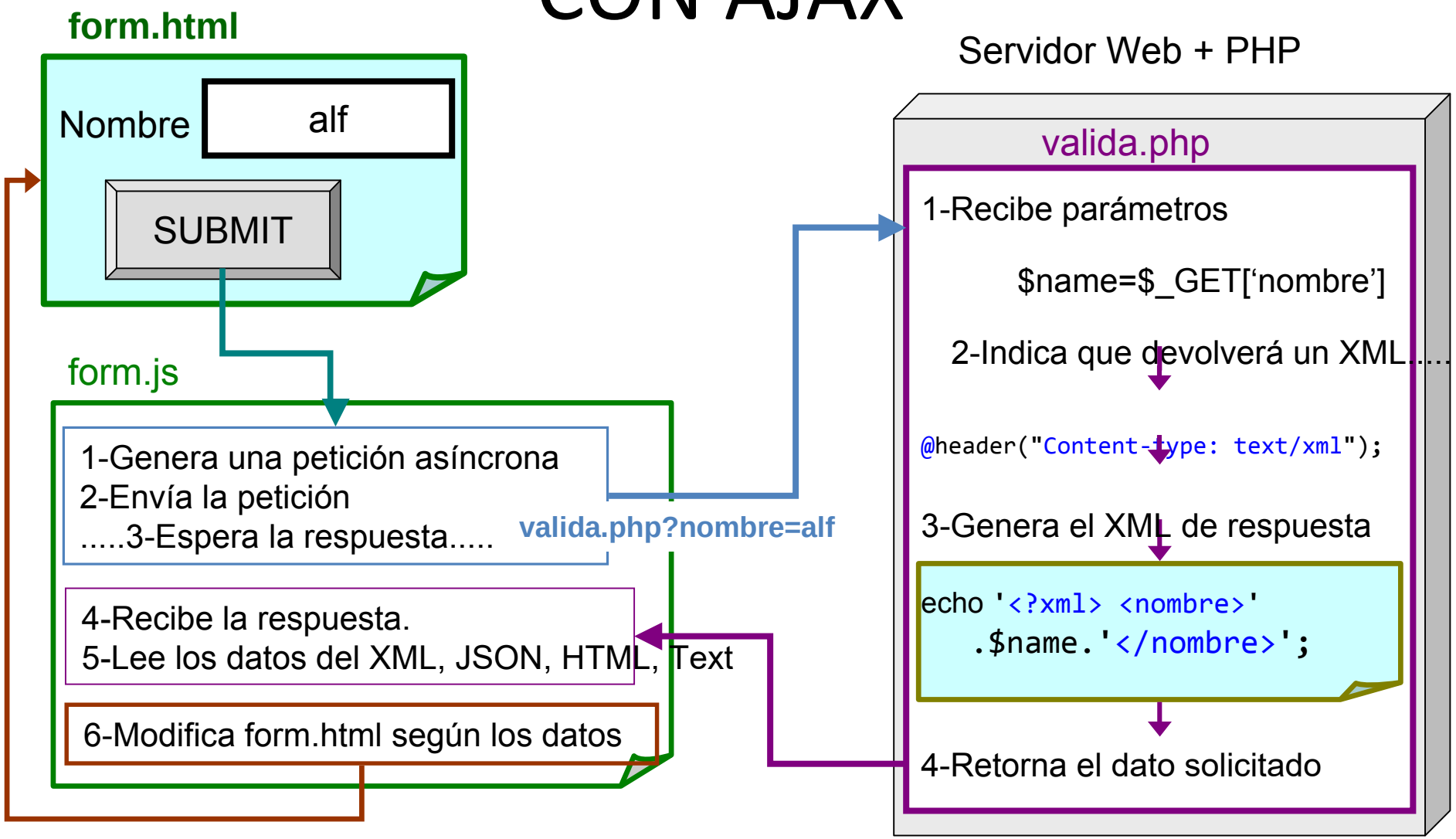
Es importante también considerar los Códigos de estado de respuesta HTTP y los estados de la petición AJAX:

Estado	Valor
•READY_STATE_UNINITIALIZED	0 : SIN INICIALIZAR
•READY_STATE_LOADING	1 : ABIERTO
•READY_STATE_LOADED	2 : CABECERAS RECIBIDAS
•READY_STATE_INTERACTIVE	3 : CARGANDO
•READY_STATE_COMPLETE	4 : COMPLETADO

PARADIGMA EN LA COMUNICACIÓN SIN AJAX



PARADIGMA EN COMUNICACIÓN CON AJAX



ActiveXObject : XMLHttpRequestn CrossBrowser

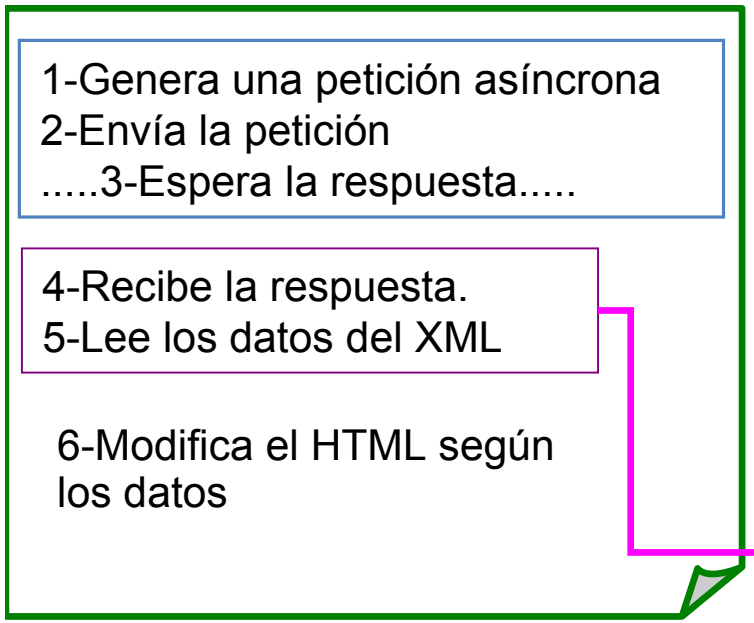
```
/**Retorna un objeto XMLHttpRequest(); */
function getXMLHttpRequest(){
var xmlHttp = null;
try{        // Internet Explorer
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
catch (e){           // Firefox, Opera 8.0+, Safari
  try{
    xmlHttp=new XMLHttpRequest();
    }
  catch (e){
    try{
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
    catch (e){
      alert("Your browser does not support AJAX!");
      return false;
      }
    }
  }
if (!xmlHttp) return null;
         return xmlHttp;
}
```

```
function enviaPeticion(){
   var xmlHttp= getXMLHttpRequest();

   var urlDestino="valida.php?nombre=alf";

   xmlHttp.open("GET", urlDestino, true);
  xmlHttp.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");

   xmlHttp.onreadystatechange = function(){
       if (xmlHttp.readyState == 4){
          funcionResp(xmlHttp);
       }
    };

    xmlHttp.send(null);
}
```

ReadyState

0:sin inicializar
1: abierto
2: cabeceras recibidas
3: cargando
4:completado

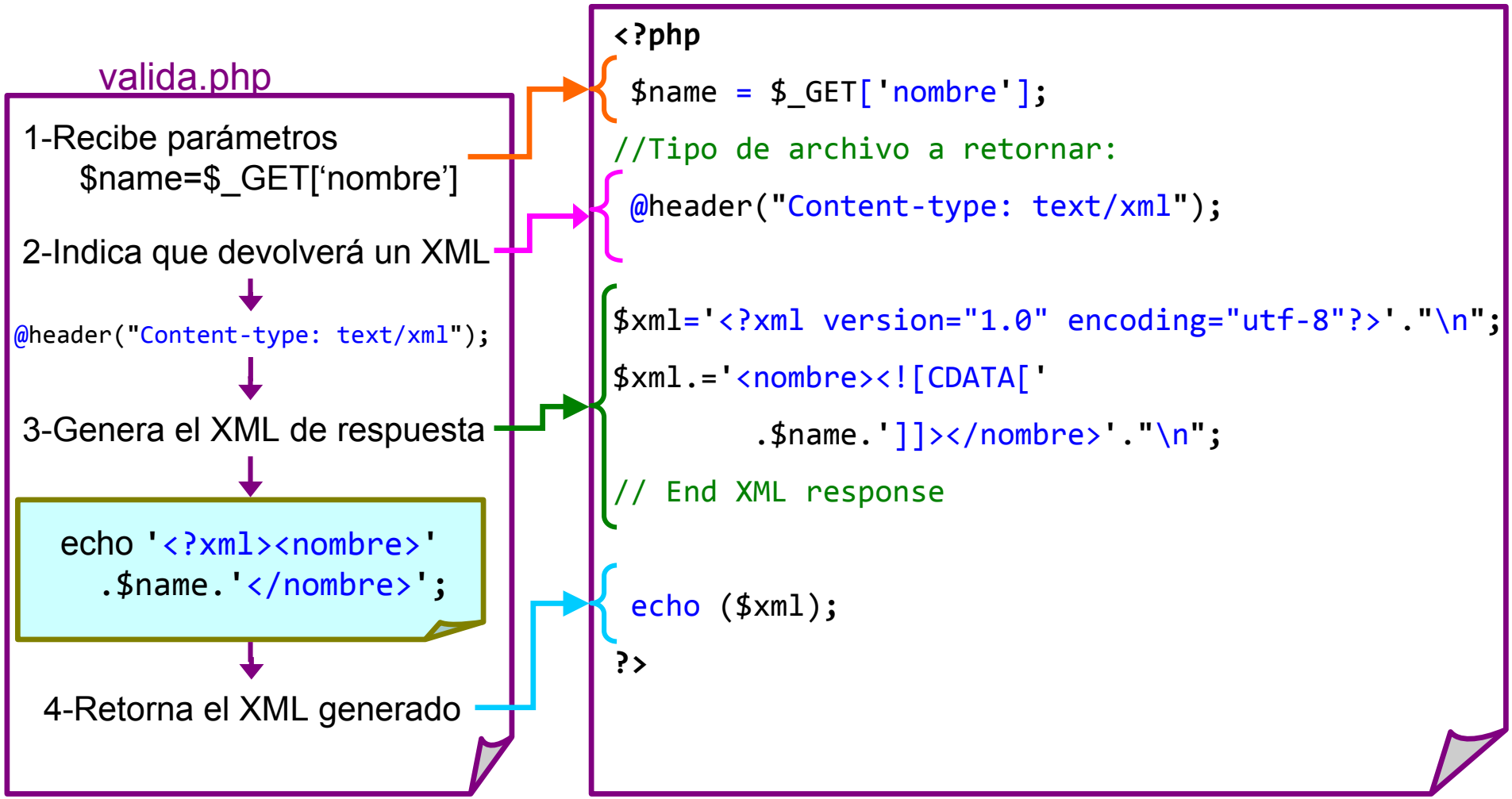
PETICION GET CON OBJETO XMLHttpRequest



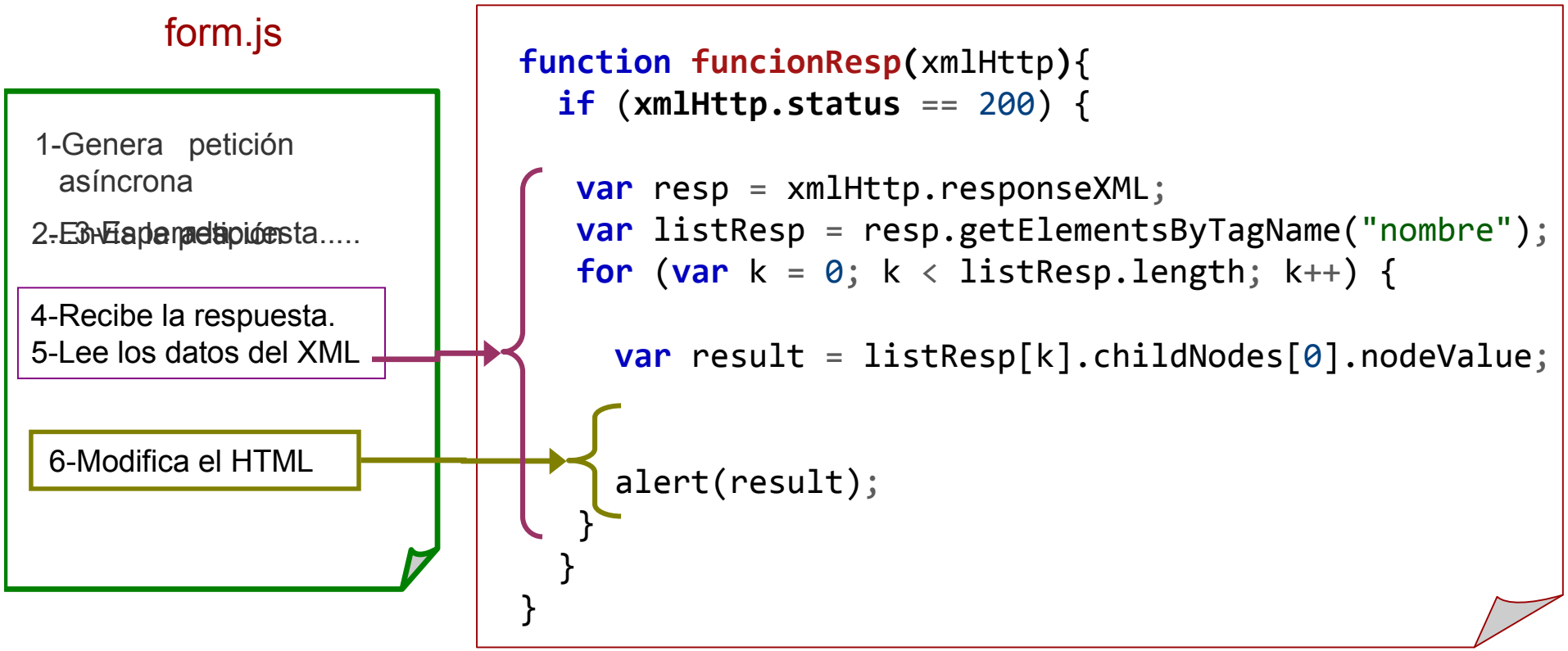

```
function enviaPeticion(){
  const xmlHttp= new XMLHttpRequest();

  const urlDestino="valida.php?nombre=alf";

 xmlHttp.open("GET", urlDestino, true);
 xmlHttp.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");

  xmlHttp.onreadystatechange = function(){
      if (xmlHttp.readyState == 4){
         funcionResp(xmlHttp);
      }
   };

   xmlHttp.send(null);
}
```

xmlHttp.open("method",url,true/false);
true=asíncrona
false=síncorna

GENERAR LA RESPUESTA XML



RECIBIR LA PETICIÓN EN FORMATO XML



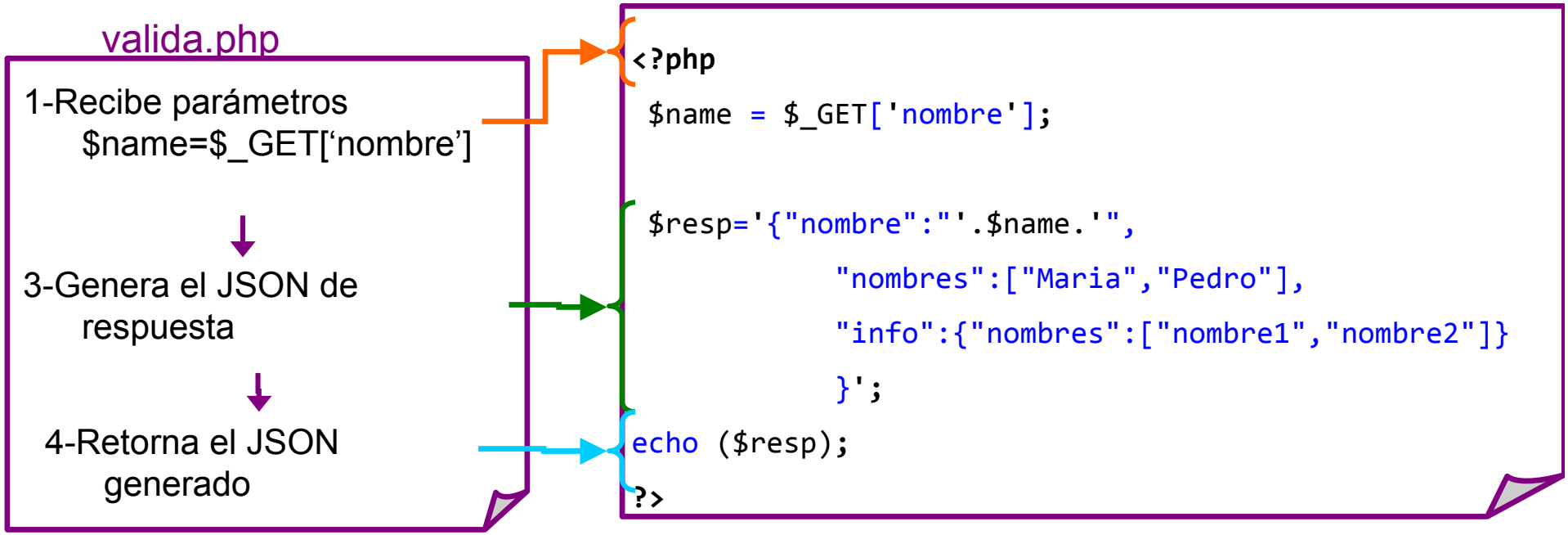
ReadyState

0:sin inicializar 1: abierto 2: cabeceras recibidas
3: cargando 4:completado

Status

200: OK
404: Page not found

GENERAR LA RESPUESTA JSON



- JSON (JavaScript Object Notation) es un lenguaje de estructuracion de datos.
- Formado por parejas : "**nombre**":"**valor**" separadas por comas.
- "**nombre**":"**valor**" es el valor de "**nombre**"
- "**nombre**":["**valor1**", "**valor2**"] es una array de 2 elementos("valor1" i "valor2").
nom[0] contiene *valor1*, *nom[1]* contiene *valor2*
- "**nombre**":{"**nom1**":"**valor1**", "**nom2**":"**valor2**"} es una *"array associativa"* de 2 elementos.
nom.nom1 contiene *valor1*, *nom.nom2* contiene *valor2*

RECIBIR LA PETICIÓN EN FORMATO JSON

form.js

1-Genera petición asíncrona
2-Envía la petición
.....3-Espera la respuesta.....

4-Recibe la respuesta.
5-Lee los datos del JSON

6-Modifica el HTML

```
function funcionResp(xmlHttp){
  if (xmlHttp.status >= 200 && xmlHttp.status <
300) {

        const resp = xmlHttp.responseText;

        let respJSON = JSON.parse(resp);

        let txt =respJSON.nombre;
        let nom1=respJSON.nombres[0]; let
        nom2=respJSON.nombres[1];
        nom1=respJSON.info.nombres[0];
        nom1=respJSON.info.nombres[1];

    }
  }
```

Estructura del JSON que leemos

```
{"nombre":"'.$name.'",
 "nombres":["Cristian","Pedro"],
 "info":{"nombres":["nombre1","nombre2"]}
}
```

PETICION POST OBJETO XMLHttpRequest

1-Genera una petición asíncrona
2-Envía la petición
.....3-Espera la respuesta.....

4-Recibe la respuesta.
5-Lee los datos del XML

6-Modifica el HTML según los datos

funcionResp() la definiremos más a delante

```
function enviaPeticion(){
   const xmlHttp= getXMLHttpRequest();

   const urlDestino="valida.php";

   xmlHttp.open("POST", urlDestino, true);
  xmlHttp.setRequestHeader("Content-Type",
'application/x-www-form-urlencoded");

   xmlHttp.onreadystatechange = function(){
      if (xmlHttp.readyState == 4){
         funcionResp(xmlHttp);
      }
    };

    xmlHttp.send(null);
}
```

API Fetch

```
fetch('./js/script.js')
   .then(function(response) {
      // si la promesa de la petición fetch se ha resuelto
      // correctamente entrará aquí.
      // sin embargo, tal vez el archivo no exista en el servidor (código 404)
      // o algún otro problema, por ello debemos de asegurarnos de que todo
      // ha ido OK (código 2XX)
      if (response.ok) {
         // response.text() devuelve otra promesa con el contenido
         // devuelto por el servidor en formato texto
         return response.text();
      } else {
         // el servidor ha respondido la petición, pero se ha producido algún
         // problema y no ha podido enviar el recurso solicitado;
         // como por ejemplo que el archivo no existe
         return Promise.reject(response.statusText);
      }
   })
   .then(function(responseText) {
      // si la promesa con el texto del script se ha resuelto
      // correctamente entrará aquí
      loadScr(responseText);
   })
   .catch(function(error) {
      // ha habido algún error al resolver alguna de las promesas
      console.log("Error al realizar la petición AJAX: " + error);
   });
```

ENVIAR ARCHIVOS CON FORMDATA

- **FormData** Es un objeto JavaScript que simula un formulario y lo podemos enviar en un XMLHttpRequest por post.
- En el **FormData** le podemos añadir los elementos del formulario HTML que queramos

Funcionamiento

- Seleccionamos el arxivo con un input type="file".
- Desde JS seleccionamos el objecto que contiene los archivos seleccionados (*control.files*).
- Añadimos cada archivo junto un nombre al FormData. (en el ejemplo solo añadimos uno)
- Finalmente creamos la petición y enviamos la información del FormData en el send().

```html
<input type="file" name="arxius" id="arxius">
<input type="button" onclick="sendFile();">
```

function sendFile()

```
function sendFile(){
    const control = document.getElementById("arxius");

    const arxiu = control.files;

    const form = new FormData();
    form.append("arxiu",arxiu[0]);

    const xmlHttp = new getXMLHttpRequest();
    xmlHttp.open("POST", 'ajax.php', true);
}   xmlHttp.send(form);
```

- En el PHP recibimos los archivos en la variable $_FILES
- Pera acceder a la información de una imagen en concreto, accedemos según el nombre indicado al añadir la imagen al FormData (en este caso *arxiu*)

saveFile.php

```php
if ($_FILES["arxiu"]["error"] > 0)
{
    $text= "Error: " . $_FILES["arxiu"]["error"] ;
}else{
    $text= "Upload: " . $_FILES["arxiu"]["name"];
    $text.="Type: " . $_FILES["arxiu"]["type"];
    $text.= "Size: " . ($_FILES["arxiu"]["size"]/1024)."kB";
    $text.= "Stored in: " . $_FILES["arxiu"]["tmp_name"];
}
```