Desarrollo de Aplicaciones Web

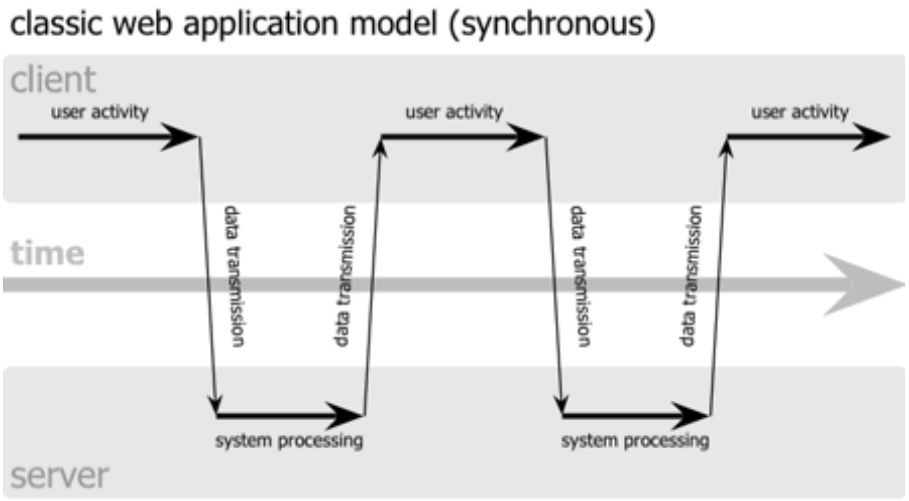
Desarrollo web en entorno cliente

Tema 5: AJAX y Promesas

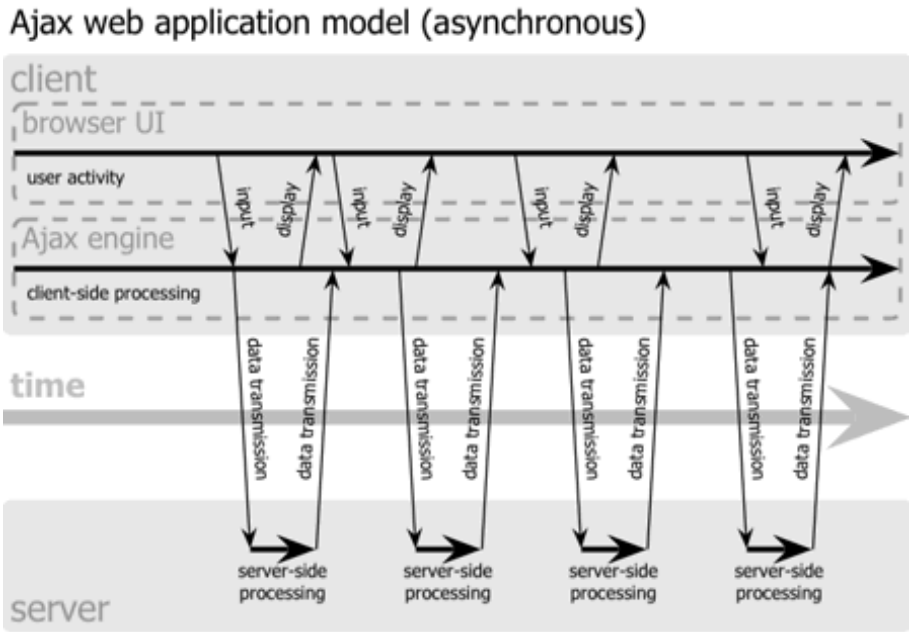
1. Ajax

AJAX (Asynchronous Javascript And XML) es un conjunto de técnicas utilizadas para hacer peticiones asíncronas, al servidor, desde JavaScript.

Al hacer una petición al servidor de forma tradicional, la página se **recarga al completo** con lo que envía el servidor cuando este proporciona una respuesta.



Con Ajax la página no se bloquea esperando respuesta, sino que continúa con su actividad normal hasta que llega la respuesta. Esos datos se reciben y procesan en segundo plano en una función creada a tal efecto, y no será necesaria la recarga de toda la página pudiéndose modificar parcialmente.



Un ejemplo claro de esto es un servidor de correo electrónico. El usuario puede estar escribiendo un mensaje y sin que se recargue la página puede ver que ha llegado uno nuevo.

En lo que a XML se refiere, es el formato en que se intercambia la información entre el servidor y el cliente, aunque actualmente el formato más usado es JSON.

1.1. Tipos de peticiones HTTP

Las peticiones Ajax usan el protocolo HTTP y tienen el formato clásico de cabeceras HTTP, tipo de petición y pudiendo incluir parámetros o datos. Algunos de los tipos de peticiones son:

- **GET**: se utiliza para obtener datos. Si se envían datos en la petición se incluyen en la URL de la petición. Es más rápido que POST y tiene límite de caracteres.
- **HEAD**: similar a GET, pero en este caso el servidor solo devolverá cabeceras, sin cuerpo de respuesta.
- **POST**: por norma se usa para enviar datos al servidor, sobre todo cuando hay una gran cantidad de datos a enviar. Los datos van en el cuerpo de la petición HTTP.
- **PUT**: similar a POST, pero suele utilizarse para actualizar datos del servidor. Estos se envían en el cuerpo de la petición y la información para identificar el objeto a modificar en la URL.
- **DELETE**: se utiliza para eliminar un dato del servidor. La información para identificar el objeto a eliminar se envía en la URL.

1.2. Respuesta a una petición HTTP

Cuando un servidor contesta a una petición HTTP, no solo devuelve los datos pedidos, si no que éstos van acompañados de cierta información proporcionada por las cabeceras.

Parte de esa información deberá recuperarse para tratar la respuesta dependiendo de ciertos datos. El código de estado, texto que acompaña a ese estado, tipo de la información que recibimos como respuesta, etc. será necesario que se tengan en cuenta para poder dar respuesta al usuario final.

2. Promesas

Las promesas permiten crear, y controlar, procesos asíncronos de una forma simplificada. La promesa es una instancia de **Promise** y, dicho objeto, representará el estado y resultado de un proceso asíncrono. Se trata de un objeto que encapsulará una operación asíncrona, ya sea una llamada Ajax, un evento, etc.

Una promesa puede estar en uno de los siguientes estados:

- **pending** (pediente): Estado inicial, implica que el proceso no está completo.
- **fulfilled** (cumplida): Una vez terminado el proceso, si todo ha ido bien.
- **rejected** (rechazada): Una vez terminado el proceso, si ha ocurrido algún error.

Al crear una instancia de Promise, se le pasará como parámetro una función que, a su vez, tendrá dos parámetros de entrada que son callbacks proporcionados por JavaScript:

- **resolve**: función callback que se ejecutará si la promesa pasa a estado fulfilled.
- **reject**: función callback que se ejecutará si la promesa pasa a estado rejected.

```
let promesa = new Promise((resolve, reject) => {
    //Cuerpo de la función para, por ejemplo, hacer una petición al servidor
    });
```

Una llamada a una promesa cuenta con algunos métodos, entre ellos:

- **.then(datos => { … }):** al resolverse la promesa satisfactoriamente se ejecuta la función pasada como parámetro del then. Ésta recibe los datos que se pasan al resolver (resolve) la promesa (normalmente los datos devueltos por la función asíncrona a la que se ha llamado).
- **.catch(datos => { … })**: la función pasada como parámetro se ejecuta si se rechaza la promesa (normalmente porque ha recibido una respuesta errónea del servidor). Esta función recibe la información pasada por la promesa al ser rechazada (reject) (normalmente información sobre el error producido).
- **.finally():** la función pasada como parámetro se ejecutará siempre, ya se haya resuelto o rechazado la promesa.

```
let promesaAsincr = new Promise(function (resolve, reject) {
   let numero = Number(prompt("Introduce un número mayor que 5"));
   if (numero > 5) {
       resolve("El número es mayor que 5");
   } else {
       reject("El número es menor que 5");
   }
});
promesaAsincr
   .then(resultado => console.log(resultado))
   .catch ( resultado => console.log(`Error ${resultado}`))
   .finally (() => alert("Proceso terminado"));
```

La llamada a la función asíncrona se haría al ejecutarse la función pasada como parámetro al crear la promesa.

3. API Fetch

La API fetch es una alternativa a XMLHttpRequest para realizar peticiones AJAX e interactuar por HTTP, basada en promesas y con mayor flexibilidad y capacidad de control a la hora de realizar llamadas al servidor. También está disponible en Node, por lo que es posible utilizarlo tanto en cliente como en servidor.

Como utiliza promesas, devuelve un objeto con los dos métodos ya conocidos, then() y catch(), a los que se pasa sendas funciones que serán las que se invoquen en caso de respuesta(respuesta es cualquier código http, incluido el que suponga error en la petición) o error(fallos de conexión por ejemplo).

Una diferencia con lo visto hasta el momento sobre promesas, es que en este caso hay que recuperar dos promesas. A continuación, se verá el por qué.

3.1. Pasos básicos

Los pasos básicos a seguir para hacer una petición son:

- Se realiza la petición a través del método fetch().
- La llamada a fetch devuelve una promesa.
- El método then() de la promesa devuelta entrega un objeto Response, que no son los datos pedidos sino la respuesta HTTP. De esta respuesta se saca el cuerpo de la misma que será devuelto como promesa también.
- El método then() de esa segunda promesa recibe el cuerpo devuelto por el servidor y será donde se traten los datos.
- Se incluirá también un catch() para tratar cualquier posible error.

```
fetch("https://jsonplaceholder.typicode.com/users/1") //devuelve una promesa
    .then(response => response.json()) //El método json() devuelve otra promesa
        .then(datosUsuario => console.log(datosUsuario)) //then() de la segunda promesa
        .catch(error => console.error(error));
```

La promesa que es capturada en el primer then(), tiene unas propiedades (status, statusText, ok, headers, ...) y unos métodos como json(). Este método devuelve otra promesa que cuando se resuelve contiene los datos JSON de la respuesta pasada.

Haciendo uso de la API Fetch, se evita toda la parte de crear la petición y gestionarla.

EJERCICIO 1

- En el Aula Virtual encontrarás el ejemplo “Petición Ajax”. En el documento peticionAjaxFecth.js del mismo, se realiza una petición GET a un servidor gratuito de pruebas:
  - Accede a ese servidor y mira que datos ofrece.
  - Ejecuta el ejemplo.
  - Mira el código del ejemplo e intenta entender cada paso.
  - Plantea las dudas que tengas.

- Identifica la estructura de los datos devueltos, una vez parseados a formato JS.
- Crea una página web que tenga un botón para desencadenar la petición del ejemplo del punto 3.1. Visualiza, en la consola, los valores que ha devuelto la petición.
- Crea otra página web que al abrirla muestre, en una tabla, todos los nombres y direcciones de correo de los usuarios (https://jsonplaceholder.typicode.com/users).

3.2. Propiedades y métodos de la respuesta

Algunas de las propiedades respuesta devuelta por fetch() son:

- **status**, **statusText**: el código y el texto del estado devuelto por el servidor (200/Ok, 404/Not found, …)
- **ok**: booleano que será true si el status es 20X y false en cualquier otro caso.
- **Headers**: Cabeceras de la respuesta.

Y los métodos que ofrece son:

- **json()**: método que devuelve una promesa con los datos JSON convertidos a un objeto (hace automáticamente JSON.parse())
- **text()**, **blob()**, **formData()**, **arrayBuffer()**, **FormData()**: Métodos que devuelven una promesa con los datos convertidos al formato correspondiente.
- **clone()**: Crea y devuelve un clon de la instancia en cuestión.
- **Response.error()**: Devuelve un nuevo objeto Response con un error de red asociado.
- **Response.redirect(url, code):** Redirige a la url indicada, con un código de error(opcional).

Siguiendo el ejemplo anterior:

```
fetch("https://jsonplaceholder.typicode.com/users/1")
    .then(response => {
          if (response.ok) { return response.json();
          } else {
          return "Error HTTP:"+response.status+"("+ response.statusText + ")";
          }
    })
        .then(datosUsuario => console.log(datosUsuario))
        .catch(error => console.error(error));
```

EJERCICIO 2

- Actualiza la solución dada al último punto del ejercicio 1, para controlar distintos estados de la respuesta.

3.3. Opciones de la petición

En los ejemplos de los puntos anteriores, no se ha indicado más que la URL a la que se estaba dirigiendo la petición. Por defecto, si no se indica lo contrario, es una petición GET.

Si se quiere enviar más información en la petición HTTP (cabeceras, tipo de petición, datos, etc) se puede incluir en un segundo parámetro (llamado init) que admite fetch(). Este parámetro adicional será un objeto JavaScript que incluirá toda la información necesaria.

Algunas de las opciones que se pueden configurar son las siguientes:

- **method**: tipo de petición que se va a realizar.
- **headers**: cabeceras que se deben enviar.
- **body**: datos a enviar al servidor, codificados en distintos formatos. Las peticiones tipo GET o HEAD no lo admiten.
- **credentials**: credenciales utilizadas: 'omit', 'same-origin', 'include'. Por ejemplo, si se quieren establecer cookies, esta opción tiene que ser establecida.
- **cache**: forma de utilización de la caché: 'default', 'no-store', 'reload', 'no-cache', 'force-cache', 'only-if-cached'.

En el siguiente ejemplo se hace una petición de tipo POST enviando los datos de un nuevo cliente:

```
let URL = "http://localhost:3000/clientes";
let nuevoUser = {
        nombre: "Diego",
        apellidos: "Garcia Rodriguez",
        DNI: "43127983D",
        fechaNac: "1977-11-02",
        Sexo: "H",
        preferencias: "ext_indv_int"
};          insertarPedido:
let init = {
        method: 'POST',
        body: JSON.stringify(nuevoUser),
        headers: { 'Content-Type': 'application/json'}
};
fetch(URL, init)
   .then(response => response.json())
     .then(datosRespuesta => alert(datosRespuesta.id))
     .catch(err => console.error(err));
```

EJERCICIO 3

- Los siguientes ejercicios van a consistir en realizar peticiones de distinta índole a la API, ya vista en el ejercicio 2, https://jsonplaceholder.typicode.com. Las peticiones que se realicen serán al recurso /todos, que contiene tareas. Las acciones a realizar son (muestra por pantalla o consola los resultados de alguna manera):
  - Realiza una petición para la que se devuelvan todas las tareas.
  - Obtén la tarea con id 55.
  - Obtén la tarea con id 201 (no existe).
  - Crea una nueva tarea. En el cuerpo de la petición se pasarán los datos a almacenar: userID=5, title="Prueba de POST", completed=false.
  - Comprueba el código de respuesta devuelto y busca su significado.
  - Modifica la tarea con id 76 para que su title sea 'Tarea modificada'. Comprueba el código de respuesta.
  - Elimina la tarea con id 32. Comprueba qué devuelve en este caso la API.

3.4. Formato de datos

EJERCICIO 4

- Siguiendo los pasos detallados en el documento "Preparar entorno para peticiones" que encontrarás en el Aula Virtual, instala la API que se indica.
- Crea una página web con un formulario que tendrá:
  - Una lista desplegable con diferentes tipos de animales.
  - Un botón para solicitar información.
  - Cuando el usuario pulse el botón, se le mostrará el nombre de todos los animales que haya de ese tipo y los rasgos de cada uno.
  - Los datos se pedirán al json-server instalado en el punto anterior.

Alguno de los formatos con los que se pueden enviar datos en una petición son:

Enviar datos en formato URIEncoded

En este caso, los datos se envían como hace el navegador por defecto en un formulario, es decir, añadiendo a la URL, tras el símbolo "?", pares nombre=valor:

?nombre=Sonia&apellido1=Gutiérrez&apellido2=García

Enviar datos en formato JSON

Otra opción es enviar los datos en formato JSON. Para ello, tras crear el objeto con los datos a enviar al servidor, es necesario pasarlos al formato JSON. Si con el método JSON.parse(datos) se convierten datos en formato JSON a un objeto JavaScript, para hacer el camino inverso se utiliza el método **JSON.stringify(datos).**

EJERCICIO 5

- Arranca el servicio animales.json.
- Crea una página web que tenga dos inputs de tipo radio ("Ver animales" y "Nuevo animal"):
  - Si se selecciona uno se realizará una petición de todos los animales que haya registrados. Se mostrará por pantalla el nombre de todos ellos.
  - Si se selecciona el otro se dará de alta uno nuevo (los datos no hace falta pedirlos, los puedes meter directamente en código).

EJERCICIO 6

- Crea un formulario en el que se puedan introducir datos de nuevos clientes de una tienda de material deportivo. Los datos a introducir son:
  - Nombre de cliente.
  - Apellidos (en un solo campo).
  - DNI.
  - Fecha de nacimiento en formato dd/mm/aaaa.
  - Sexo, radio buttons (entre paréntesis lo que se envíe al servidor):
    - Mujer (M).
    - Hombre (H).

- Preferencias, que serán checkbox, con las opciones (los valores entre paréntesis será lo que se envíe al servidor y se almacenarán en el mismo campo separados por _):
  - Deportes de exterior (ext).
  - Deportes de equipo (equi).
  - Deportes individuales (indv).
  - Deportes de interior (int).

- Fijándote en el documento animales.json, crea un clientes.json. Al final de este ejercicio puedes ver un ejemplo.
- La página web tendrá, además, una tabla con el siguiente formato:

Id	Nombre y apellidos	Fecha nacimiento	Sexo
1	Sonia Gutiérrez García	10/10/1010	M

- Cada vez que se introduzca un nuevo cliente, se enviarán los datos al servidor mediante una petición asíncrona (el id no ha de mandarse, lo creará la API automáticamente), en formato JSON.
- Cuando llegue la respuesta a la petición, se incluirá una nueva fila en la tabla con los datos del cliente dado de alta.
- Una vez terminados todos los puntos anteriores:
  - Añade un botón al formulario de tal manera que, cuando se pulse, lo que se haga sea borrar el usuario con el DNI introducido. El resto de campos no hace falta que se introduzcan porque no se van a necesitar.
  - Añade otro botón para actualizar los datos de un cliente ya dado de alta. Con el DNI del cliente, se podrán modificar cualquiera de los datos.
- Ejemplo del contenido de clientes.json:

```
{
        "clientes": [
          {
                "id": 1,
                "nombre": "Sonia",
                "apellidos": "Gutiérrez García",
                "DNI": "45893125t",
                "fechaNac": "10/10/1010",
                "Sexo": "M",
                "preferencias": "ext_equi"
          }
        ]
}
```