UD7. Programación AJAX en JavaScript

DWEC
Mª Luz Sánchez Rubio

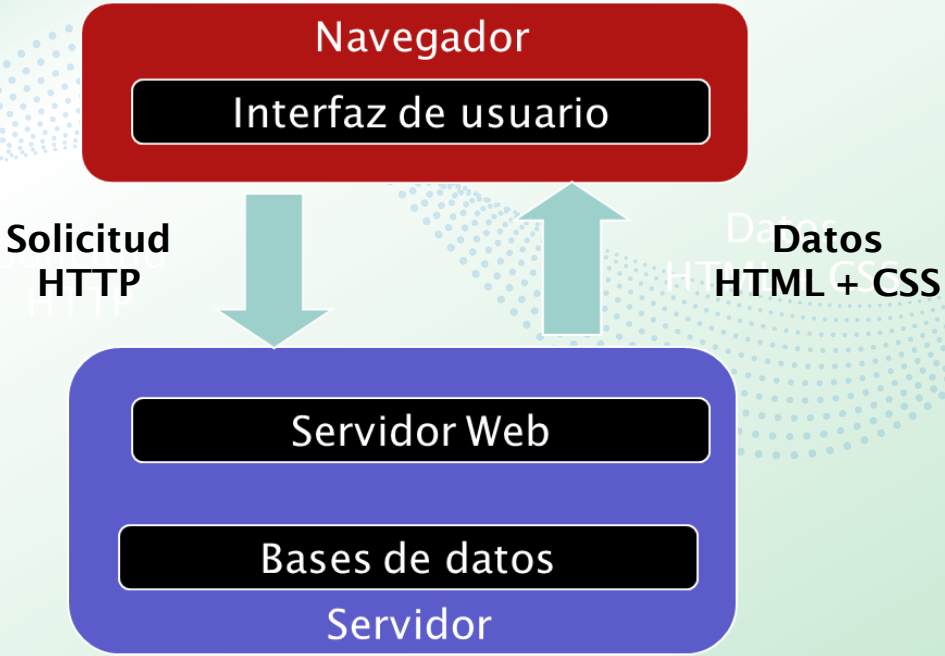
OBJETIVOS

- Conocer los mecanismos de comunicación asíncrona en las aplicaciones web.
- Conocer las tecnologías asociadas con la técnica y su utilización en el desarrollo de aplicaciones interactivas.
- Profundizar en los formatos de envío y recepción de información asíncrona.
- Conocer en detalle la realización de llamadas asíncronas.
- Describir las librerías de actualización dinámicas actuales.

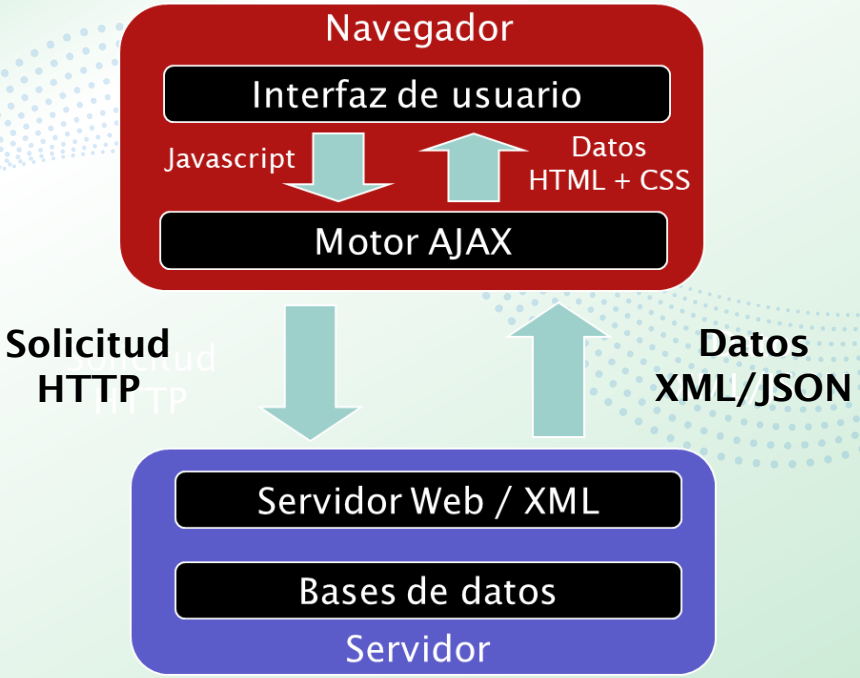
1. ¿Qué es AJAX?

- Tecnología para realizar solicitudes asíncronas al servidor.
- Actualiza partes de una página web sin recargar toda la página.
- Acrónimo: **Asynchronous JavaScript and XML**. (aunque hoy en día también usa JSON).
- Basado en JavaScript y respuestas HTTP.
- AJAX no es un nuevo lenguaje de programación, sino una nueva forma de usar las normas existentes.
- Base de aplicaciones web modernas.

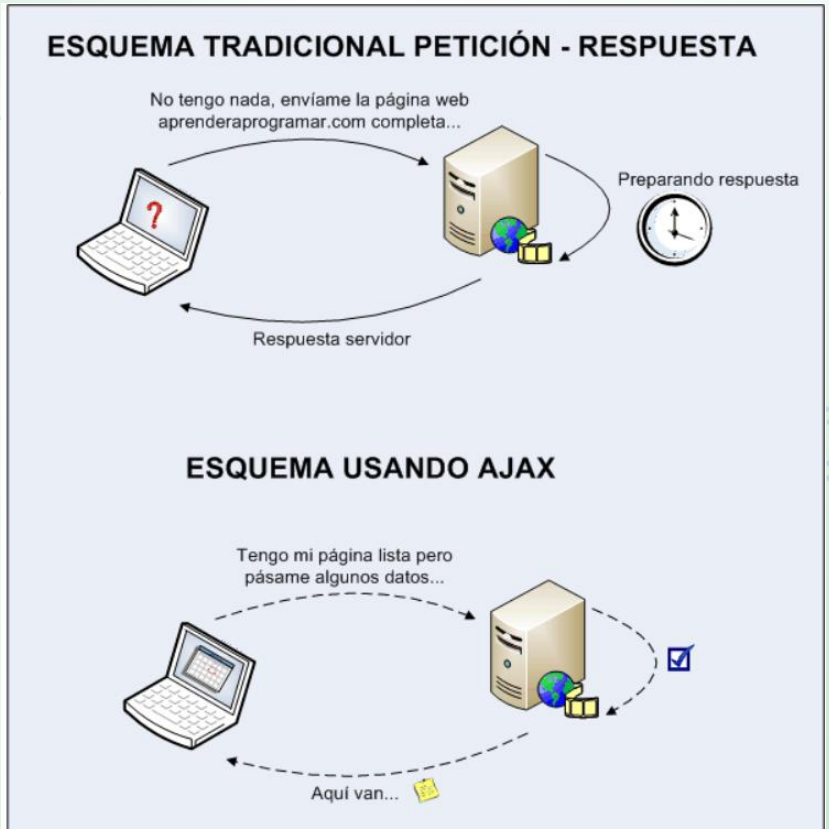
2. Módelo clásico de aplicaciones web



2. Módelo AJAX de aplicaciones web



2. Módelo clásico & AJAX



3. Beneficios de AJAX

- Mejora la experiencia del usuario.
- Reducción de carga del servidor.
- Incremento en la velocidad de las páginas web.
- Compatible con múltiples formatos de datos (JSON, XML, HTML, texto).

4. Aplicaciones Web que usan AJAX

- **Motores de búsqueda y autocompletado**.
  - **Google Search**: Solicitudes en tiempo real para sugerir resultados mientras el usuario escribe.
- **Aplicaciones de redes sociales**.
  - **Facebook, Twitter**: Actualización de feeds sin recargar la página y publicación de comentarios y "**Me gusta**" en tiempo real.
- **Comercio electrónico.**
  - **Amazon, MercadoLibre:** Actualización dinámica de carritos de compra. Filtros de búsqueda sin refrescar la página.

4. Aplicaciones Web que usan AJAX

- **Aplicaciones de mapas**.
  - **Google Maps:** Carga de mapas y resultados de direcciones sin recarga.
- **Aplicaciones mensajería.**
  - **WhatsApp Web**: Envío y recepción de mensajes sin recargar la página y notificaciones en tiempo real.

5. JSON vs - XML

- En el mundo del desarrollo web y la transferencia de datos, dos formatos han destacado **JSON** (**J**ava**S**cript **O**bject **N**otation) y **XML** (e**X**tensible **M**arkup **L**anguage).
- Ambos son muy utilizados para estructurar y representar información.

json

```json
{
  "nombre": "Juan",
  "edad": 30,
  "hobbies": ["leer", "viajar"]
}
```

xml

```xml
<persona>
  <nombre>Juan</nombre>
  <edad>30</edad>
  <hobbies>
    <hobby>leer</hobby>
    <hobby>viajar</hobby>
  </hobbies>
</persona>
```

5. JSON vs - XML

JSON es un formato de intercambio de datos ligero y fácil de entender.

Ventajas

1. Sintaxis concisa.
2. Ligero.
3. Nativa compatible con JS.
4. Procesamiento eficiente.

Desventajas

1. Menos descriptivo
2. Falta de comentarios estructurales

5. JSON vs - XML

XML es un lenguaje de marcas que permite definir etiquetas personalizadas para estructurar datos. Ha perdido terreno frente a JSON en aplicaciones web.

Ventajas

1. Flexibilidad.
2. Descriptivo.
3. Validación.

Desventajas

1. Documentos más grandes y difíciles de leer.
2. Procesamiento más lento.
3. Tamaño de archivo mayor.

6. Métodos para implementar AJAX

- **XMLHttpRequest (XHR):** El método tradicional.
- **jQuery AJAX:** Método popular en proyectos con JQuery.
- **Fetch API:** Alternativa moderna.
- **Axios:** Librería ligera basada en Promises.

7. El objeto XMLHttpRequest

- Proporciona la capacidad de interactuar con servidores web de manera asíncrona.
- Creado originalmente para manejar solicitudes de XML, pero soporta otros formatos (JSON, HTML, etc).
- Manejo de solicitudes en los métodos **GET** y **POST.**
- **Creación de XHR:**

  *const **xhr** = new **XMLHttpRequest()**;*

- **Métodos básicos de XHR:**
  - **open():** Configura la solicitud (método, URL, asincronía).
  - **send()**: Envía la solicitud.
  - **setRequestHeader():** Define encabezados HTTP para los métodos POST.

7. El objeto XMLHttpRequest

- **Método GET con XMLHttpRequest**
  - ✓ Se utiliza para recuperar datos del servidor.
  - ✓ Los parámetros se envían en la URL.

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data?param1=value1&param2=value2', true);

xhr.onreadystatechange = () =>{
   if (xhr.readyState === 4 && xhr.status === 200) { //La solicitud ha sido completada y respuesta
                                                     //exitosa
      console.log(JSON.parse(xhr.responseText));
   }
};
xhr.send();
```

7. El objeto XMLHttpRequest

- **Método POST con XMLHttpRequest**
  - ✓ Se utiliza para recuperar datos del servidor.
  - ✓ Los parámetros se envían en la URL.

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://api.example.com/submit', true);
// Establecer el encabezado HTTP 'Content-Type' para especificar que los datos enviados son en formato JSON
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onreadystatechange = ()=> {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(JSON.parse(xhr.responseText));
    }
};
const data = JSON.stringify({ param1: 'value1', param2: 'value2' });
xhr.send(data);
```

7. El objeto XMLHttpRequest

- **Uso con promesas:**

```
function getData(url) {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.open('GET', url, true);
        xhr.onload = () => {
            if (xhr.status === 200) {
                resolve(JSON.parse(xhr.responseText));
            } else {
                reject(new Error(`Error ${xhr.status}: ${xhr.statusText}`));
            }
        };
        xhr.onerror = () => reject(new Error('Error de red'));
        xhr.send();
    });
}
```

7. El objeto XMLHttpRequest

- **Uso con promesas:**

```
getData('https://api.example.com/data')
    .then(data => console.log(data))
    .catch(error => console.error(error));
```

- **Uso con async/await**

```
async function fetchData() {
  try {
    const data = await getData('https://api.example.com/data');
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
//llamada a la función
fetchData();
```

8. AJAX con jQuery

- Simplificó el uso de AJAX con métodos más legibles y concisos.
- Flexibilidad con configuraciones personalizadas
- Potente herramienta para crear experiencias dinámicas
- Soporte para promesas y configuraciones avanzadas.
- **.done()**: Se ejecuta cuando la solicitud es exitosa.
- **.fail()**: Se ejecuta cuando la solicitud falla.

8. AJAX con jQuery

- **Métodos en jquery**:

1. $.**get**() y $.**pos**t()
   - Métodos simplificados para solicitudes GET y POST.
2. $.**getJSON**()
   - Simplificación para obtener datos en formato JSON.
3. $.**load()**
   - Carga contenido HTML directamente en un elemento del DOM.
4. $.**ajaxSetup**()
   - Configuración predeterminada para todas las solicitudes AJAX.
5. $.**ajax()**
   - Método general para realizar peticiones AJAX. Permite alta personalización.

8. AJAX con jQuery

- **Ejemplos de métodos en jquery**:

  - $.**get**(). Solicitudes GET rápidas

```
$.get('url', { clave: 'valor' }, (data)=> {
  console.log(data);
});
```

  - $.**pos**t(). Manejo de datos en JSON.

```
$.post('url', function(data) {
  console.log(data);
});
```

8. AJAX con jQuery

- **Ejemplos de métodos en jquery**:

  - $.**load**(). Inserta directamente contenido HTML en DOM

    *$('#miElemento').**load**('url ');*

  - $.**getJSON**(). Manejo de datos en JSON

    ```
    $.getJSON('url', function(data) {
      console.log(data);
    });
    ```

8. AJAX con jQuery

- **Ejemplos de métodos en jquery**:
  - $.**ajax**(). Método general para realizar peticiones AJAX. Permite alta personalización.

```
$.ajax({
    url: 'https://jsonplaceholder.typicode.com/posts',
    method: 'GET',
   data:{
         nombre: ‘Luis’
       }
})
.done(function(response) {    //se ejecuta si la solicitud es exitosa
  console.log('Datos recibidos:', response);
  $('#resultado').text(`Título: ${response.title}`);
})
.fail(function(error) { //se ejecute si ocurre un error durante la solicitud
  console.error('Ocurrió un error:', error);
  $('#resultado').text('Error al cargar los datos.');
});
```

8. AJAX con jQuery

- **Con async/await**

```
// Función asíncrona que realiza la solicitud AJAX
const obtenerDatos = async () => {
  try {
    const data = await $.ajax({
      url: 'https://jsonplaceholder.typicode.com/posts',
      method: 'GET',
      data:{
            nombre: ‘Luis’
            }
    });
    console.log('Datos recibidos:', data);
    $('#resultado').text(`Título: ${data.title}`);
  } catch (error) {
    console.error('Ocurrió un error:', error);
    $('#resultado').text('Error al cargar los datos.');
  }
}
// Llamar a la función
obtenerDatos();
```

9. Fetch

- **fetch()** es una API de JavaScript que permite realizar solicitudes HTTP desde del navegador.
- Permite obtener datos de un servidor o enviar datos a él de manera asíncrona.
- Basado en promesas.
- Más fácil de usar que XMLHttpRequest.
- Soporta operaciones GET, POST, PUT, DELETE, etc.

9. Fetch

- **GET (Obtener datos):** Es el método por defecto.

```
fetch('https://api.example.com/data')
  .then(response => response.json()) // Convierte la respuesta a JSON
  .then(data => console.log(data))    // Muestra los datos
  .catch(error => console.error('Error:', error)); // Manejo de errores
```

9. Fetch

- **POST (Enviar datos):** Por ejemplo, con formato JSON.

```
fetch('https://api.example.com/submit', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ key: 'value' })
});
```

9. Fetch

- **Manejo de errores:** fetch() solo rechaza promesas por fallos de red o con problemas de CORS. No rechaza por respuestas HTTP no exitosas (como 404 o 500)

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

10. Axios

- **Axios** es una librería para hacer peticiones HTTP en JavaScript.
- Está basada en promesas, lo que facilita el manejo de respuestas asincrónicas.
- Se usa conmunmente para interactuar con APIs RESTful.
- Compatible con navegadores y Node.js

10. Axios

- La instalación de Axios puede realizarse con:
  - **npm** o **yarn** en tu proyecto
  - Incluyendo la librería en el navegador con script

10. Axios

- **Sintaxis con GET**:

```
axios.get('https://api.example.com/data?ID=12345')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.log(error);
  });
```

10. Axios

- **Sintaxis con GET con params**:

```
axios.get('https://api.example.com/data', {
        params: {
                ID: 12345
        })
 .then(response => {
   console.log(response.data);
 })
 .catch(error => {
   console.log(error);
 });
```

10. Axios

- **Sintaxis con GET con async/await**

```
const getExample =async () =>{
   try {
      const response = await axios.get ('https://api.example.com/data?ID=12345');
      console.log(response.data);
    } catch (error) =>{
      console.log(error);
    });
```

10. Axios

- **Sintaxis básica con POST**:

```
axios.post('https://api.example.com/data', { key: 'value' })
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.log(error);
  });
```

11. Explorando los diferentes tipos de Content-Type en HTTP

- El *Content-Type* (tipo de contenido) es un encabezado HTTP utilizado para indicar el tipo de datos que se están enviando en el cuerpo de la solicitud o respuesta.
- El servidor utiliza este encabezado para saber cómo procesar los datos que se le envían.
- Tipos comunes de Content-Type:
  - **application/json**: Datos en formato json. Usado cuando se necesitan enviar datos complejos o estructurados, como objetos o arrays, en un API/Restful.
  - **application/x-www-form-unlencoded**: Datos codificados en URL (formulario clásico).
  - **multipart/form-data:** Usado para enviar archivos como imágenes, vídeos o documentos.
  - **text/plain**: Usado para enviar datos sencillos en texto sin formato.

11. Explorando los diferentes tipos de Content-Type en HTTP

- **Ejemplos**:

```
//application/json
const xhr = new XMLHttpRequest();
xhr.open("POST", "https://api.example.com/data", true);
xhr.setRequestHeader("Content-Type", "application/json");

// application/x-www-form-urlencoded
fetch('https://api.example.com/submit', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  },
  body: 'name=John&age=30'
})
```

11. Explorando los diferentes tipos de Content-Type en HTTP

- **Ejemplos**:

//text/plain

```
axios.post('https://api.example.com/message', 'Hello, World!', {
headers: { 'Content-Type': 'text/plain' }
})
```

//multipart/form-data

```
const formData = new FormData();
formData.append('id', document.querySelector('i#dUser').value);
formData.append('username', 'JohnDoe');

// Realizar la solicitud POST con Fetch
fetch('https://api.example.com/upload', {
  method: 'POST',
  body: formData  // El cuerpo de la solicitud es un objeto FormData
})
```