Recuerden poner a grabar la clase

Clase 20

Ajax

Diplomatura UNTREF

Temario

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

AJAX:

- Fundamentos de la tecnología AJAX
- Evolución - Peticiones HTTP
- Estados de una petición: Documentación
- Thunder Client
- Interactuar con un servidor de Backend
- URL Params
- Query Params
- Peticiones Fetch()

Fundamentos de la tecnología AJAX

Fundamento de la tecnología AJAX

AJAX, que significa Asynchronous JavaScript and XML (JavaScript y XML Asíncronos), es una tecnología que se utiliza en el desarrollo web para crear aplicaciones web interactivas y dinámicas. Fue popularizado a principios de la década de 2000 y ha sido ampliamente adoptado desde entonces.

Asincronía en AJAX

La característica clave de AJAX es la capacidad de realizar solicitudes al servidor web de forma asincrónica.

Esto significa que una página web puede enviar y recibir datos del servidor en segundo plano sin necesidad de recargar toda la página.

La asincronía permite que las aplicaciones web sean más receptivas y evita que el usuario tenga que esperar a que se cargue toda la página cuando se realiza una acción.

"Ejemplo de asincronía en AJAX sería la carga de comentarios en una publicación de redes sociales. Cuando un usuario hace clic en "ver comentarios", se envía una solicitud al servidor en segundo plano para recuperar los comentarios sin tener que recargar toda la página."

JavaScript y AJAX

AJAX se basa en JavaScript para realizar la comunicación con el servidor y actualizar el contenido de la página de manera dinámica.

JavaScript es un lenguaje de programación que se ejecuta en el navegador web del cliente, lo que permite que las interacciones del usuario y las solicitudes al servidor ocurran de manera fluida sin requerir una recarga completa de la página.

XMLHttpRequest AJAX

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

XMLHttpRequest es el objeto fundamental en JavaScript que permite realizar solicitudes HTTP asincrónicas.

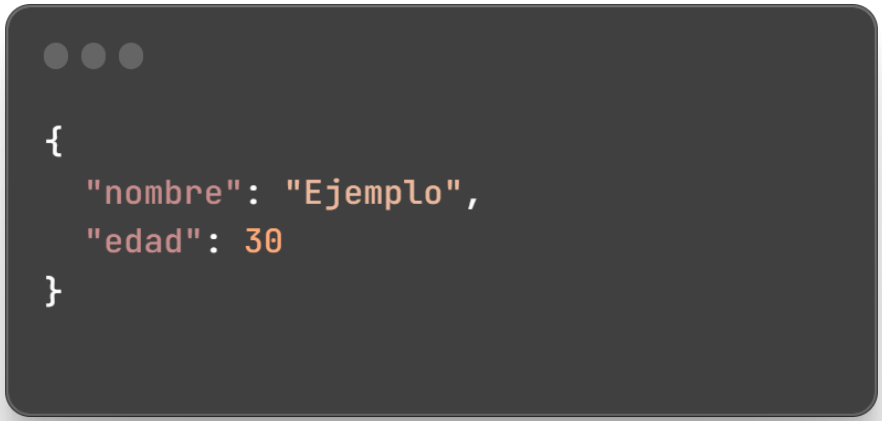
A través de este objeto, puedes enviar peticiones al servidor web, recibir respuestas y manipular los datos recibidos.

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://ejemplo.com/api/datos", true);
xhr.onreadystatechange = function () {
  if (xhr.readyState == 4 && xhr.status == 200) {
    let respuesta = JSON.parse(xhr.responseText);
    console.log(respuesta);
  }
};
xhr.send();
```

Formato de datos AJAX

Aunque el nombre de **AJAX** incluye **"XML" (Extensible Markup Language)**, en la práctica, los datos intercambiados entre el cliente y el servidor pueden estar en varios formatos, como **XML, JSON (JavaScript Object Notation)** o incluso texto plano.

JSON es el formato más comúnmente utilizado debido a su simplicidad y facilidad de manipulación en JavaScript.



Manipulación del DOM en AJAX

Una vez que se reciben los datos del servidor, se pueden utilizar para actualizar el contenido de la página web en tiempo real.

Esto permite a las aplicaciones web mostrar información nueva o modificar elementos existentes en la página sin necesidad de recargarla por completo.

La manipulación del **DOM (Document Object Model)** es una parte esencial de la programación AJAX.

Gestiones de errores y estados AJAX

Es importante gestionar adecuadamente los errores durante las solicitudes AJAX y mantener un seguimiento de los estados de las solicitudes, como la carga, el éxito o el fracaso.

Esto garantiza una experiencia de usuario más sólida y robusta.

```
if (xhr.readyState == 4) {
  if (xhr.status == 200) {
    // Procesar respuesta exitosa
  } else {
    // Manejar error
  }
}
```

Seguridad en AJAX

Al utilizar **AJAX**, es esencial tomar medidas de seguridad para protegerse contra ataques como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF).

Validar y sanear los datos recibidos del cliente, autenticar y autorizar las solicitudes adecuadamente son prácticas importantes para garantizar la seguridad de una aplicación web.

"Por ejemplo, puedes utilizar token CSRF en tus solicitudes para proteger contra CSRF y asegurarte de que los datos ingresados por el usuario se validen adecuadamente antes de procesarlos."

Evolucion - Peticiones HTTP

Seguridad en AJAX

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

AJAX (Asynchronous JavaScript and XML) es una técnica que permite realizar peticiones a un servidor web desde una página web sin necesidad de recargar toda la página.

Esto permite crear aplicaciones web más interactivas y dinámicas, ya que los datos pueden ser cargados y actualizados en segundo plano sin interrumpir la experiencia del usuario.

Evolucion y peticiones en HTTP

La evolución de las peticiones HTTP en AJAX ha pasado por varias etapas:

XMLHttpRequest: Esta fue la primera API utilizada para realizar peticiones HTTP asincrónicas en JavaScript.

Fue introducida por Microsoft en Internet Explorer 5 en la década de 1990 y posteriormente adoptada por otros navegadores.

Con XMLHttpRequest, los desarrolladores podían enviar solicitudes HTTP al servidor y recibir respuestas sin recargar la página.

Sin embargo, tenía limitaciones y carecía de ciertas características modernas.

Evolucion y peticiones en HTTP

JSON y mejor manejo de datos:

A medida que AJAX se popularizaba, la comunidad comenzó a utilizar **JSON (JavaScript Object Notation)** en lugar de XML para el intercambio de datos. JSON es más ligero y fácil de analizar en JavaScript, lo que lo convirtió en la elección preferida para el intercambio de datos.

Además, se mejoró la forma en que los datos se manejaban en las respuestas de las solicitudes, lo que facilitó su manipulación.

Evolucion y peticiones en HTTP

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Librerías y frameworks AJAX:

A medida que las aplicaciones web se volvían más complejas, surgieron librerías y frameworks que simplificaban el uso de AJAX. Algunos ejemplos incluyen jQuery, Axios, Fetch API y AngularJS.

Estas herramientas proporcionaron abstracciones y funcionalidades adicionales para facilitar la realización de solicitudes HTTP y el manejo de respuestas.

Evolucion y peticiones en HTTP

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Promesas y async/await:

Con la introducción de las promesas en JavaScript y posteriormente la sintaxis async/await, el manejo de las solicitudes AJAX se volvió más limpio y fácil de entender.

Estas características permiten escribir código asincrónico de manera más clara y evitar el anidamiento excesivo de callbacks.

Evolucion y peticiones en HTTP

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Fetch API:

La Fetch API es una API moderna para realizar peticiones HTTP en JavaScript que reemplazó en gran medida a XMLHttpRequest.

Fetch proporciona una interfaz más flexible y basada en promesas para realizar solicitudes y manejar respuestas.

Se ha convertido en la elección preferida para hacer peticiones AJAX en aplicaciones web modernas.

Estados de una peticion

Estado de una petición

Los estados de una petición HTTP son una parte fundamental del ciclo de vida de una solicitud realizada desde un cliente (como un navegador web) a un servidor web.

En general, se utilizan cinco estados principales para describir el progreso de una solicitud HTTP:

```
// URL de la API a la que haremos la solicitud
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

// Crear una solicitud usando fetch
const request = fetch(apiUrl);

// Monitorear los estados de la solicitud
request
  .then((response) => {
    if (response.status === 200) {
      // La solicitud se ha completado exitosamente
      console.log('Solicitud completada con éxito');
      return response.json(); // Obtener los datos de la respuesta en formato JSON
    } else {
      // La solicitud ha tenido un error
      console.error('Error en la solicitud:', response.status, response.statusText);
    }
  })
  .then((data) => {
    // Manejar los datos de la respuesta (si la solicitud tuvo éxito)
    console.log('Datos recibidos:', data);
  })
  .catch((error) => {
    // Manejar errores de red u otros errores
    console.error('Ocurrió un error en la solicitud:', error);
  });
```

Estado de una petición

Pendiente (Pending): Este es el estado inicial de una solicitud HTTP.
Ocurre cuando se ha iniciado una solicitud, pero aún no se ha enviado al servidor o se está esperando una respuesta. En este estado, la solicitud está en proceso de ser configurada y enviada.

Enviado (Sent): Una vez que la solicitud ha sido completamente configurada y enviada al servidor, entra en el estado "Enviado".
En este punto, la solicitud ha salido del cliente y se encuentra en tránsito hacia el servidor. El servidor aún no ha respondido.

```
// URL de la API a la que haremos la solicitud
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

// Crear una solicitud usando fetch
const request = fetch(apiUrl);

// Monitorear los estados de la solicitud
request
  .then((response) => {
    if (response.status === 200) {
      // La solicitud se ha completado exitosamente
      console.log('Solicitud completada con éxito');
      return response.json(); // Obtener los datos de la respuesta en formato JSON
    } else {
      // La solicitud ha tenido un error
      console.error('Error en la solicitud:', response.status, response.statusText);
    }
  })
  .then((data) => {
    // Manejar los datos de la respuesta (si la solicitud tuvo éxito)
    console.log('Datos recibidos:', data);
  })
  .catch((error) => {
    // Manejar errores de red u otros errores
    console.error('Ocurrió un error en la solicitud:', error);
  });
```

Estado de una petición

Recibido (Received): Cuando el servidor recibe la solicitud y comienza a procesarla, la solicitud entra en el estado "Recibido".
En este punto, el servidor está procesando la solicitud y preparándose para enviar una respuesta.

Completado (Completed): Una vez que el servidor ha procesado la solicitud y ha generado una respuesta, la solicitud pasa al estado "Completado".
En este estado, el servidor ha respondido y la respuesta se encuentra disponible para su procesamiento en el cliente.

```
// URL de la API a la que haremos la solicitud
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

// Crear una solicitud usando fetch
const request = fetch(apiUrl);

// Monitorear los estados de la solicitud
request
  .then((response) => {
    if (response.status === 200) {
      // La solicitud se ha completado exitosamente
      console.log('Solicitud completada con éxito');
      return response.json(); // Obtener los datos de la respuesta en formato JSON
    } else {
      // La solicitud ha tenido un error
      console.error('Error en la solicitud:', response.status, response.statusText);
    }
  })
  .then((data) => {
    // Manejar los datos de la respuesta (si la solicitud tuvo éxito)
    console.log('Datos recibidos:', data);
  })
  .catch((error) => {
    // Manejar errores de red u otros errores
    console.error('Ocurrió un error en la solicitud:', error);
  });
```

Estado de una petición

Error (Error): Si ocurre algún problema durante la solicitud, como un error de red, una URL incorrecta, un error del servidor, etc., la solicitud puede entrar en el estado "Error".

En este caso, no se recibe una respuesta válida o completa.

```
// URL de la API a la que haremos la solicitud
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';

// Crear una solicitud usando fetch
const request = fetch(apiUrl);

// Monitorear los estados de la solicitud
request
  .then((response) => {
    if (response.status === 200) {
      // La solicitud se ha completado exitosamente
      console.log('Solicitud completada con éxito');
      return response.json(); // Obtener los datos de la respuesta en formato JSON
    } else {
      // La solicitud ha tenido un error
      console.error('Error en la solicitud:', response.status, response.statusText);
    }
  })
  .then((data) => {
    // Manejar los datos de la respuesta (si la solicitud tuvo éxito)
    console.log('Datos recibidos:', data);
  })
  .catch((error) => {
    // Manejar errores de red u otros errores
    console.error('Ocurrió un error en la solicitud:', error);
  });
```

Estado de una petición

Es importante destacar que estos estados pueden variar según la implementación y la tecnología utilizada.

Por ejemplo, cuando se realizan solicitudes AJAX en JavaScript, se pueden usar eventos para monitorear estos estados y ejecutar acciones específicas en cada uno de ellos.

Thunder Client

Thunder Client

En general, las extensiones para navegadores web que están diseñadas para probar API REST suelen proporcionar una interfaz gráfica de usuario que permite a los desarrolladores realizar solicitudes HTTP a endpoints de API y ver las respuestas de manera fácil y conveniente.

Estado de una petición

Creación de solicitudes: Permiten especificar el método HTTP (GET, POST, PUT, DELETE, etc.), los encabezados, los parámetros y el cuerpo de la solicitud de una manera fácil de usar.

Historial de solicitudes: Almacenan un historial de las solicitudes previamente realizadas, lo que facilita la repetición de solicitudes anteriores.

Estado de una petición

Pruebas y validación: Permiten agregar pruebas y validaciones a las solicitudes para asegurarse de que las respuestas cumplan con ciertos criterios.

Colecciones: A menudo permiten organizar las solicitudes en colecciones para una mejor gestión.

Variables y entornos: A menudo, estas extensiones permiten definir variables y entornos para simplificar la configuración de solicitudes.

Interactuar con un servidor BackEnd

interactuar con servidor BackEnd

Interactuar con un servidor de backend es una parte fundamental en el desarrollo de aplicaciones web y móviles.

Los servidores de backend son responsables de gestionar la lógica de negocio, la base de datos y proporcionar datos o servicios a las aplicaciones cliente (como aplicaciones web, aplicaciones móviles, etc.).

Algunos puntos claves a tener en cuenta son:

interactuar con servidor BackEnd

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Realizar solicitudes HTTP: La forma más común de interactuar con un servidor de backend es a través de solicitudes HTTP. Puedes usar métodos **HTTP** como **GET** (para obtener datos), **POST** (para enviar datos al servidor), **PUT** (para actualizar datos), **DELETE** (para eliminar datos), entre otros.

Puedes realizar estas solicitudes desde el lado del cliente utilizando JavaScript en una aplicación web o desde un lenguaje de programación en una aplicación móvil.

interactuar con servidor BackEnd

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

URLs y rutas: Debes conocer las URLs o rutas de los endpoints en el servidor de backend que deseas acceder. Estas rutas especifican qué recursos o servicios se están solicitando. Por ejemplo, /usuarios podría ser una ruta para obtener una lista de usuarios.

Envío de datos: Para enviar datos al servidor, normalmente utilizas el cuerpo de la solicitud **HTTP.** Puedes enviar datos en formato **JSON**, formularios HTML o cualquier otro formato requerido por la API del servidor.

interactuar con servidor BackEnd

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Encabezados HTTP: A menudo, es necesario incluir encabezados HTTP en tus solicitudes para autenticación, configuración o para especificar el formato de respuesta deseado. Los encabezados son clave-valor que se envían junto con la solicitud.

Autenticación y autorización: Dependiendo de la API del servidor, es posible que necesites autenticarte para acceder a ciertos recursos. Esto se logra generalmente enviando credenciales (como tokens de acceso) en los encabezados de la solicitud.

Interactuar con servidor BackEnd

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Recibir respuestas: Cuando realizas una solicitud al servidor, este responde con una respuesta HTTP. La respuesta puede contener datos, códigos de estado **(como 200 OK o 404 Not Found)**, y otros metadatos. Debes procesar la respuesta en tu aplicación cliente de acuerdo a tus necesidades.

Manejar errores: Si algo sale mal en la solicitud **(por ejemplo, el servidor está caído, la ruta no existe, hay un error en el servidor, etc.),** debes manejar estos errores adecuadamente en tu aplicación para proporcionar una experiencia de usuario robusta.

Url Params

URL Params

Los **"URL params" o "parámetros de URL"** se refieren a valores que se pasan dentro de la URL de una solicitud HTTP. Estos parámetros son utilizados para enviar información adicional al servidor web, generalmente para personalizar una solicitud o para filtrar los resultados que el servidor debe devolver.

Los parámetros de URL generalmente siguen el formato **nombre=valor** y se agregan a la URL después del signo de interrogación **?**.
Puedes tener múltiples parámetros en una URL, y se separan entre sí con el símbolo **&**.

Query Params

Query Params

Los "query params" o "parámetros de consulta" en JavaScript se utilizan para extraer y trabajar con los parámetros de consulta (query parameters) de una URL. Estos parámetros son los que siguen al signo de interrogación (?) en una URL y se utilizan para enviar datos adicionales en una solicitud HTTP GET.

Por ejemplo, en la URL **https://ejemplo.com/buscar?termino=perro&color=marron**, los parámetros de consulta son termino y color.

Query Params

Para trabajar con parámetros de consulta en JavaScript, puedes utilizar el objeto **URLSearchParams** si estás trabajando en un entorno de navegador web.

Aquí tienes un ejemplo de cómo usarlo:

```
// Obtener la URL actual del navegador
const url = new URL(window.location.href);

// Obtener un parámetro de consulta específico
const termino = url.searchParams.get("termino");
console.log(`Término de búsqueda: ${termino}`);

const color = url.searchParams.get("color");
console.log(`Color: ${color}`);

// Verificar si un parámetro existe
if (url.searchParams.has("edad")) {
  const edad = url.searchParams.get("edad");
  console.log(`Edad: ${edad}`);
} else {
  console.log("El parámetro 'edad' no está presente en la URL.");
}

// Obtener todos los parámetros de consulta
url.searchParams.forEach((valor, nombre) => {
  console.log(`${nombre}: ${valor}`);
});
```

¿Dudas o consultas?

¡Muchas Gracias!