JS SEMINARIO DE LENGUAJES

EXPLICACIÓN PRÁCTICA 3

# INTERACTIVIDAD Y DATOS EN TIEMPO REAL CON JAVASCRIPT

# AGENDA

1. Serialización De Objetos
2. Web Apis Sobre Http
3. Uso De Apis Externas Y Ajax
4. Introducción A Websockets

# SERIALIZACIÓN DE OBJETOS

La serialización es el proceso de convertir un objeto en un formato que puede ser fácilmente transmitido o almacenado, permitiendo su reconstrucción posterior en un ambiente diferente.

# JSON

JSON (JavaScript Object Notation) es un formato de texto ligero de intercambio de datos que es fácil de leer y escribir para los humanos y fácil de analizar y generar para el software.

```
const objeto = {
  torneo: "Copa de la Liga",
  campeon: "Estudiantes de La Plata",
  edicion: 2024,
}
console.log(JSON.stringify(objeto)) // '{"torneo":"Copa de la Liga",
console.log(typeof JSON.stringify(objeto)) // 'string'
```

# JSON.STRINGIFY()

El método `JSON.stringify()` convierte un objeto o valor de JavaScript en una cadena de texto JSON.

```
console.log(JSON.stringify(1)) // '1'
console.log(JSON.stringify({equipo: "Estudiantes"})) // '{"equipo":"
console.log(JSON.stringify([2024, "false", false])) // '[2024,"false
```

# USO DE JSON.PARSE()

Después de recibir datos en formato JSON, frecuentemente necesitamos convertirlos de nuevo a un objeto para manipularlos en JavaScript.

```
const jsonString = '{"torneo":"Copa de la Liga","campeon":"Estudiant
const objeto = JSON.parse(jsonString)
console.log(objeto.campeon) // 'Estudiantes de La Plata'
```

# ARCHIVOS JSON

JSON no solo se utiliza para intercambiar datos en tiempo real entre aplicaciones web, sino que también es ampliamente utilizado como formato para almacenar datos en archivos.

Estos archivos JSON son útiles para configuraciones, intercambio de información entre diferentes programas y almacenamiento de datos estructurados de manera ligera y fácilmente accesible.

# WEB APIS SOBRE HTTP

Son interfaces de programación que permiten a las aplicaciones web interactuar con otros servicios y aplicaciones a través de la web, utilizando el protocolo HTTP como medio de comunicación.

# ¿QUÉ ES UN ENDPOINT?

Es una URL específica en una API donde se puede acceder a determinadas funciones del servidor. Estas funciones pueden incluir recuperar, enviar, actualizar o eliminar datos.

Los endpoints determinan las operaciones disponibles y cómo acceder a ellas a través de la API.

## NO PERSISTENCIA DE CONEXIÓN

Las Web APIs operan sin mantener una conexión persistente con el servidor. Esto significa que cada solicitud es tratada de forma independiente, sin retener ninguna información de las solicitudes anteriores.

# FLUJO DE UNA PETICIÓN

Una petición inicia por un cliente, éste es procesado por un servidor, devuelve una respuesta y la pretición concluye.

El servidor nunca inicia una petición por si solo.

# USO DE APIS EXTERNAS Y AJAX

AJAX (Asynchronous JavaScript and XML) es un conjunto de tecnologías permite que un usuario de la aplicación web interactúe con una página sin la interrupción que implica volver a cargar la página web.

El intercambio de información se hace de manera asincrónica haciendo llamadas a APIs.

# REALIZANDO UNA PETICIÓN AJAX CON FETCH API

La Fetch API permite hacer peticiones asincrónicas para obtener información y tomar acciones.

```
fetch("https://pokeapi.co/api/v2/pokemon/25/")
.then(response => response.json())
.then(data => console.log(data.name)) // 'Pikachu'
.catch(error => console.error('Error:', error))
```

Este ejemplo muestra cómo realizar una petición GET para obtener datos JSON desde una API externa.

# MANEJO DE RESPUESTAS Y ERRORES EN AJAX

Es crucial manejar correctamente las respuestas y posibles errores en las peticiones AJAX para asegurar la robustez de la aplicación web.

```
fetch("https://pokeapi.co/api/v2/pokemon/25/")
.then(response => {
  if (!response.ok)
    throw new Error('Network response was not ok')
  return response.json()
})
.then(data => console.log(data.name)) // 'Pikachu'
.catch(error => console.error('Error:', error))
```

# MANIPULACIÓN DEL DOM CON AJAX

Utilizar AJAX para cargar contenido de manera asincrónica permite a los desarrolladores actualizar partes de una página web sin recargar toda la página, mejorando la experiencia del usuario y la eficiencia de la aplicación.

```
fetch('https://pokeapi.co/api/v2/type/electric/')
.then(response => response.json())
.then(data => {
  const list = document.getElementById('pokeList')
  list.innerHTML = ''
  data.pokemon.forEach(item => {
    const li = document.createElement('li');
    li.textContent = item.pokemon.name
    list.appendChild(li)
  })
```

```
})
.catch(error => console.error('Error loading pokemons:', error))
```

# ACTUALIZACIÓN DE DATOS EN TIEMPO REAL

El uso de AJAX para actualizar datos en tiempo real es crucial en aplicaciones web dinámicas como dashboards o aplicaciones de redes sociales.

```
setInterval(() => {
  fetch('https://api.example.com/latest-news')
    .then(response => response.json())
    .then(data => {
      const newsContainer = document.getElementById('news')
      newsContainer.textContent = data.news
    })
}, 10000)
```

# INTRODUCCIÓN A WEBSOCKETS

Es un protocolo de comunicación que proporciona canales de comunicación bidireccionales y persistentes sobre una única conexión.

Es especialmente útil para aplicaciones que requieren intercambios frecuentes y rápidos de datos, como juegos en línea, aplicaciones de chat, y plataformas de trading en tiempo real.

# WEBSOCKETS VS. HTTP

WebSockets y HTTP son dos protocolos diferentes que se utilizan para diferentes propósitos.

HTTP es un protocolo de solicitud-respuesta, mientras que WebSockets permite una comunicación bidireccional y persistente.

# ESTABLECIMIENTO DE UNA CONEXIÓN WEBSOCKET

Establecer una conexión WebSocket implica un handshake entre el cliente y el servidor, inicializado desde el cliente. Una vez establecida, la conexión permanece abierta hasta que es cerrada por el cliente o el servidor.

```
const socket = new WebSocket('ws://www.example.com/socketserver')
socket.onopen = function(event) {
  console.log('Conexión WebSocket abierta')
}
```

# ENVÍO Y RECEPCIÓN DE MENSAJES

Una vez abierta la conexión WebSocket, tanto el cliente como el servidor pueden enviar y recibir mensajes libremente. Los mensajes pueden ser datos de texto o binarios.

```
const resultado = {
  equipo1: {
    nombre: "Estudiantes de La Plata",
    goles: 1,
  },
  equipo2: {
    nombre: "Vélez Sarsfield",
    goles: 1,
  }
}
socket.onmessage = function(event) {
  console.log('Mensaje recibido: ' + event.data)
```

```
}
```

# CIERRE DE UNA CONEXIÓN WEBSOCKET

El cierre de una conexión WebSocket puede ser iniciado por el cliente o el servidor y debe ser manejado cuidadosamente para asegurar que no haya pérdida de datos y que todos los recursos sean liberados adecuadamente.

```
socket.onclose = function(event) {
  console.log('Conexión WebSocket cerrada', event.reason)
}
socket.close()
```

# MANEJO DE ERRORES EN WEBSOCKETS

El manejo de errores es crucial en el uso de WebSockets para asegurar que la aplicación pueda responder adecuadamente a problemas como interrupciones de red o errores de comunicación.

```
socket.onerror = function(event) {
  console.error('Error en WebSocket', event)
}
```

# FIN DE LA CLASE