INSTITUTO KHIPU

Semestre IV

Sesión 27

PROGRAMA DE ESTUDIOS

DESARROLLOS DE SISTEMAS DE INFORMACIÓN

DESARROLLO FRONTEND DE SISTEMAS DE WEB

Tema

USO DE PROMESAS CON JAVASCRIPT Y JQUERY

Uso de Promesas con JavaScript y jQuery

Objetivo:

Comprender el uso de **promesas** en JavaScript y su aplicación con **jQuery** para manejar peticiones asíncronas de forma más limpia y eficiente.

🧠 PARTE CONCEPTUAL

❓ Pregunta detonadora:

¿Cómo podemos asegurarnos de que una petición al servidor se complete antes de ejecutar otra acción?

🧵 ¿Qué es una promesa?

Una promesa es un **objeto que representa la eventual finalización o fallo de una operación asíncrona**.

📌 Estados de una promesa:

- `pending` (pendiente)
- `fulfilled` (resuelta correctamente)
- `rejected` (rechazada con error)

🧱 Sintaxis básica:

```
const promesa = new Promise((resolve, reject) => {
  // operación asíncrona
  if (todoOk) resolve("Éxito");
  else reject("Error");
});
```

Encadenamiento:

```
promesa
  .then(resultado => console.log("Resultado:", resultado))
  .catch(error => console.error("Error:", error))
  .finally(() => console.log("Finalizado"));
```

🚀 Métodos útiles:

- **`.then()`** → para manejar éxito.
- **`.catch()`** → para errores.
- **`.finally()`** → se ejecuta siempre.
- **`Promise.all()`** → espera que todas las promesas se cumplan.
- **`Promise.race()`** → resuelve/rechaza con la primera que termine.

👩‍🏫 PARTE DEMOSTRATIVA (20 minutos)

1️⃣ Promesa básica

```
function tareaAsincrona() {
  return new Promise(resolve => {
    setTimeout(() => resolve("Listo en 3 segundos"), 3000);
  });
}

tareaAsincrona().then(msg => console.log(msg));
```

2 Petición con `fetch()`

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.json())
  .then(data => console.log("Título:", data.title))
  .catch(err => console.error("Error:", err));
```

3 Varias promesas con `Promise.all()`

```
Promise.all([
  fetch("https://jsonplaceholder.typicode.com/users/1").then(res =>
res.json()),
  fetch("https://jsonplaceholder.typicode.com/posts/1").then(res =>
res.json())
]).then(([usuario, post]) => {
  console.log("Usuario:", usuario.name);
  console.log("Post:", post.title);
});
```

4 Uso de promesas con jQuery

```
$.ajax({
  url: "https://jsonplaceholder.typicode.com/users/1",
  method: "GET"
})
  .done(function (data) {
    console.log("Usuario (jQuery):", data.name);
  })
  .fail(function () {
    console.log("Error en la petición AJAX con jQuery");
  });
```

✍️ ACTIVIDAD PRÁCTICA INDIVIDUAL (20 minutos)

🛠️ Enunciado:

Desarrolla un script que:

- Cree una promesa que se resuelva después de 2 segundos y muestre un mensaje.
- Use `fetch()` para traer los datos de un usuario y mostrar su email en la consola.
- Use `Promise.all()` para obtener un usuario y un post.
- Realice una solicitud AJAX con jQuery (`$.ajax`) y muestre el nombre del usuario.

💡 APIs sugeridas:

- https://jsonplaceholder.typicode.com/users/1
- https://jsonplaceholder.typicode.com/posts/1

📓 Conclusión

💬 Reflexión:

¿Cómo mejoran las promesas la estructura del código en comparación con callbacks anidados?

Posibles respuestas:

- Código más legible
- Evita la pirámide del infierno (callback hell)
- Mejor manejo de errores
- Composición más clara con `Promise.all()`

📋 Revisión de ejercicios:

- ¿Funciona el retraso con la promesa?

- ¿Se muestran los datos del `fetch()`?
- ¿`Promise.all()` retorna correctamente ambos valores?
- ¿`$.ajax()` gestiona la respuesta y el error?

🏠 Tarea para la casa:

- Crear una función en JavaScript que haga dos peticiones a diferentes APIs usando `Promise.all()` y muestre ambos resultados en pantalla.
- Crear una solicitud AJAX con `$.ajax()` que muestre un listado de usuarios en una lista HTML.

INSTITUTO

KHIPU