SEMINARIO DE LENGUAJES

CLASE 6

ASINCRONISMO EN JAVASCRIPT

Gestión asincrónica mediante callbacks, promesas y async/await

AGENDA

1. Multitasking En Javascript1
2. Clientes Y Servidores1
3. Funciones Sincrónicas1
4. Funciones Asincrónicas1
5. Objeto Promise O Promesa
6. Async/await1
7. Ventajas De Las Promesas
8. Referencias

MULTITASKING EN JAVASCRIPT[1]

¿Es posible ejecutar varios procesos simultaneamente en JavaScript?

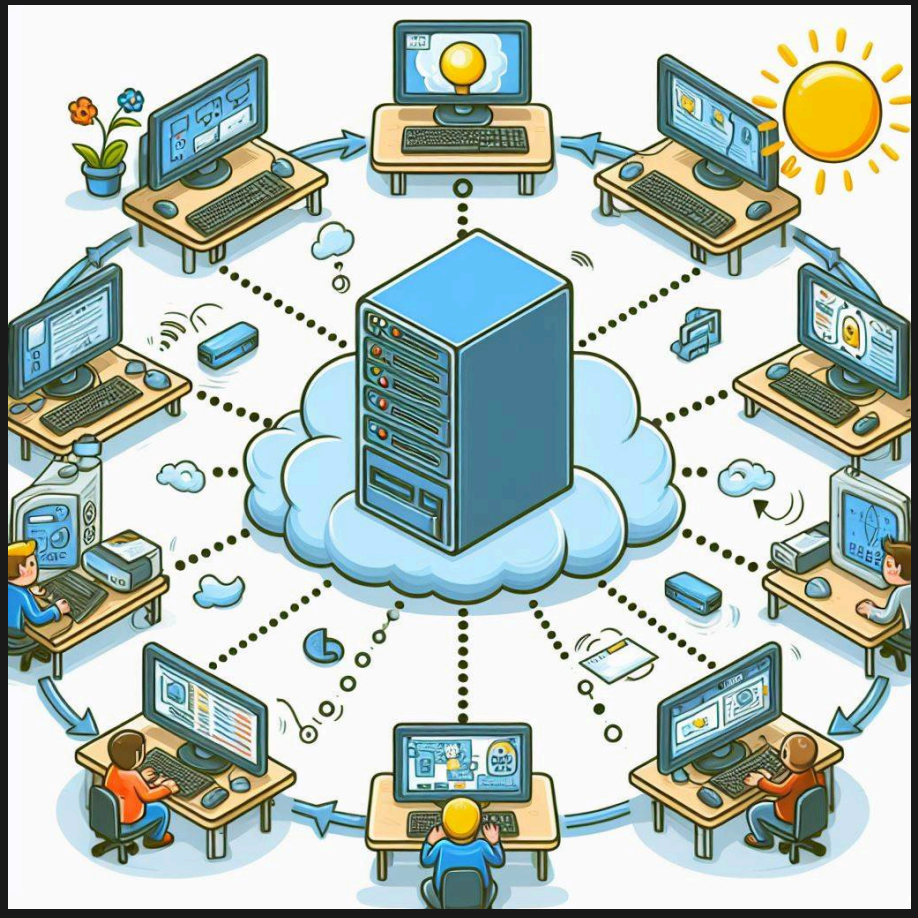
El objetivo es que si un proceso tarda no bloquee al resto de las tareas.

MULTITASKING EN JAVASCRIPT[2]

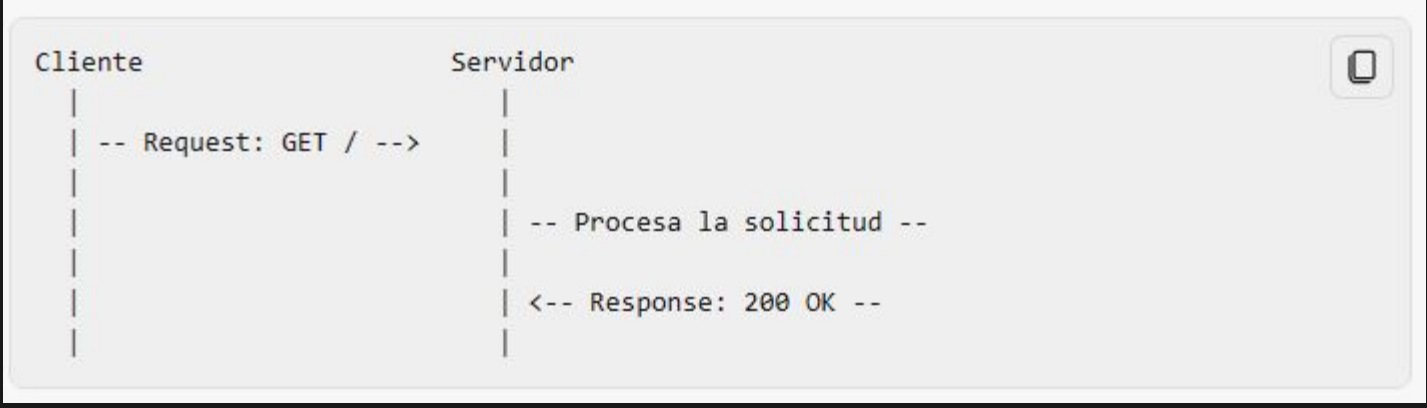
Sería similar a un mozo. Atiende más de una mesa a la vez mientras gestiona los pedidos con la cocina que se encarga de prepararlos.
Para esto implementa funciones callback.
Sigue atendiendo al cliente mientras espera los resultados del proceso en el servidor.

CLIENTES Y SERVIDORES[1]

Recordemos que cuando visitamos un sitio web con nuestro navegador predilecto (Firefox, Chrome, Edge, etc.) se produce una conexión vía Internet entre un cliente —quien inicia la conexión y solicita el contenido del sitio web— y un servidor —quien recibe la conexión y envía el contenido solicitado. Los servidores serían los negocios y los clientes quienes hacen las peticiones.



CLIENTES Y SERVIDORES[2]



El cliente realiza una petición HTTP GET al servidor pidiendo recursos como una página Web o datos. El servidor recibe la solicitud y la procesa y envía una respuesta al cliente (en esta caso 200 OK) y retorna las datos solicitados. Ver ejemplo de client.js y server.js

CLIENTES Y SERVIDORES[3]

El cliente se conecta con el DNS para localizar la dirección del servidor (la dirección de la tienda)

El navegador envía un mensaje HTTP GET al servidor pidiéndole que envíe una copia de la página Web del cliente, vía TCP/IP. Si el servidor aprueba la solicitud del cliente, envía “200 OK” y comienza a enviar los archivos de la solicitud, en trozos pequeños llamados “paquetes de datos”.

El navegador reúne todos esos trozos, forma el sitio Web completo y lo muestra (tus nuevas compras de la tienda están en casa).

CLIENTES Y SERVIDORES[4]

¿QUÉ ES HTTP Y HTTPS? | Aprende que es http, https y para que utiliza en menos 5 minutos! Y repasamos la próxima clase... Kahoot

FUNCIONES SINCRÓNICAS[1]

JavaScript tiene un hilo de proceso único, lo que significa que dos porciones del script no pueden ejecutarse al mismo tiempo, deben hacerlo una después de la otra

FUNCIONES SINCRÓNICAS[2]

```
function saludar(name) {
  return `Hola, mi nombre es ${name}!`;
}

const name ='Trinidad';
const saludo = saludar(name);
console.log(saludo);
```

saludar() espera a que se termine de ejecutar para continuar con el resto del código

FUNCIONES ASINCRÓNICAS[1]

En algunos casos se necesita ejecutar una función que lleva un tiempo, por ejemplo solicitar el contenido de un archivo, de una base de datos o hacer una petición a un sitio Web. Pueden tener un pequeño delay.
Mientras se ejecuta continúa ejecutando otros eventos.
Al finalizar ser notificado del resultado.

Ver ejemplo de Generador de primos

FUNCIONES ASINCRÓNICAS EN JS[2]

Es posible no bloquear el programa usando funciones callback.

Una función callback se usa como parámetro de un método o función. Ejemplo sentencia foreach()

Se ejecutará el tiempo estipulado por el método o función que lo utiliza.

Ver ejemplo de setTimeout

FUNCIONES ASINCRÓNICAS EN JS[3]

Ejemplos

- fetch() para realizar requerimientos HTTP
- getUserMedia() para acceder a la cámara o micrófono del usuario
- showOpenFilePicker() para permitir al usuario subir un archivo

API XMLHTTPREQUEST

Inicialmente se usaba la API XMLHttpRequest para asociar eventos asincrónicos, con un manejador de eventos.

Es un mecanismo muy confuso y con algunos problemas en la gestión de las funciones callbacks.

OBJETO PROMISE O PROMESA

Imaginemos que somos un cantante de rock y nuestros fans preguntan día y noche cuando va a estar nuestra próxima canción. Generás una lista para que dejen ahí su email y les avisarás cuando esté la canción, para que todos la reciban cuando la canción esté disponible. Si algo sale mal, un incendio en el estudio por ejemplo, también le vas a avisar. Nadie sale abrumado y los fanáticos no se pierden la canción.

OBJETO PROMISE[1]

Es un objeto de JS para manejar el asincronismo.
I Promise a Result!
"Producing code" is code that can take some time.
"Consuming code" is code that must wait for the result.
A Promise is an Object that links Producing code and Consuming code"

JavaScript Promise. w3schools

OBJETO PROMISE²

```
1  let promise = new Promise(function(resolve, reject) {
2    // Ejecutor (el código productor, "cantante")
3  });
```

La función se denomina ejecutor. La función recibe dos parámetros, objeto resolve y objeto reject. La función ejecutor corre e intenta realizar una tarea.
Cuando termina con el intento, llama a resolve si fue exitoso o reject si hubo un error.
resolve y reject son funciones callbacks proporcionadas por JS.

OBJETO PROMISE[3]

La clase Promise usa una función callback de la forma
callback(resolve, reject)
como un parámetro del constructor.

- Cuando se llama a resolve() se dispara el método .then(callback)
- Cuando se llama a reject() se dispara el método .catch(callback)

OBJETO PROMISE. PROPIEDADES

- state: inicialmente "pendiente", luego cambia a "cumplido" cuando se llama a resolve o "rechazado" cuando se llama a reject.
- result: inicialmente undefined, luego cambia a valor cuando se llama a resolve(valor) o error cuando se llama a reject(error).

Estos estados se acceden con los métodos .then/.catch/.finally

OBJETO PROMISE. MÉTODOS

OBJETO PROMISE. EJEMPLOS[1]

```
// data es un arreglo con datos.
function getData() {
  return new Promise((resolve, reject) => {
    if(data.length === 0){
      reject(new Error('data vacío'));
    }
    setTimeout(() => {
    resolve(data);
    }, 2000)
  })
  }

  getData()
    .then((response) => console.log(response))
    .catch((err) =>console.log(err.message))
```

Ver ejemplo completo de Promesas

ASYNC/AWAIT[1]

Opción más legible para escribir código asincrónico. La palabra clave async se utiliza para declarar una función asíncrona, y await se utiliza para esperar a que una promesa se resuelva antes de continuar con la ejecución del código.

ASYNC/AWAIT^2

```
async function getSeminarios(){
  const seminarios = await getData();
  console.log(seminarios);
  }

  getSeminarios();
```

Notar que la función getSeminarios parece una función sincrónica en su estructura.

await reemplaza al .then() y .catch()

Ver ejemplo completo de async/await

VENTAJAS DE LAS PROMESAS

- Legibilidad: sintaxis más limpia y fácil de entender
- Encadenamiento de promesas: facilita el encadenamiento a través de multiples then()
- Control de flujo asincrónico.
- Estados definidos: pendiente, resuelta o rechazada
- Manejo de errores: más sencillo con .catch()
- Función estandard: soportada por ECMAScript, garantiza la interoperabilidad con otras bibliotecas.

REFERENCIAS

JavaScript from Frontend to Backend. Eric Sarrion. Ed. Packt Publishing (2022)

MDN Guía de JavaScript. Usar promesas

JavaScript Promises. W3Schools

Callbacks y Promises | Explicado con ejemplos

Así funcionan las PROMESAS y ASYNC/AWAIT en JAVASCRIPT.