Clase #2: Qué es el asincronismo 2/21 🤹🏾‍♂️

✍🏾 Conceptos importantes para entender el asincronismo:

• 🧵 Thread: Thread para Javascript permite realizar programación multihilos en este entorno. En realidad, simula la creación y ejecución de hilos, pero para el desarrollador es lo mismo. Ésto simplifica muchísimo la creación de aplicaciones Javascript.

• 🚫 Bloqueante: Una llamada u operación bloqueante no devuelve el control a la aplicación hasta que se ha completado. Por tanto el thread queda bloqueado en estado de espera.

• 🚿 No bloqueante: Una tarea no bloqueante se devuelve inmediatamente con independencia del resultado. Si se completó, devuelve los datos. Si no, un error.

• 🎞️ Síncrono: Las tareas se ejecutan de forma secuencial, se debe esperar a que se complete para continuar con la siguiente tarea.

• 🚦 Asíncrono: Las tareas pueden ser realizadas más tarde, lo que hace posible que una respuesta sea procesada en diferido. La finalización de la operación I/O (entrada/salida) se señaliza más tarde, mediante un mecanismo específico como por ejemplo un callback, una promesa o un evento, lo que hace posible que la respuesta sea procesada en diferido.

• 🛤️ Paralelismo: El paralelismo es la ejecución simultánea de dos o más tareas. Algunas tareas se pueden dividir en partes más pequeñas que pueden ser resueltas simultáneamente.

• 🎮 Concurrencia: La concurrencia es la capacidad de un algoritmo o programa para ejecutar más de una tarea a la vez. El concepto es similar al procesamiento paralelo, pero con la posibilidad de que muchos trabajos independientes hagan diferentes cosas a la vez en lugar de ejecutar el mismo trabajo.

• 🌀 Eventloop o Loop de eventos: El bucle de eventos es un patrón de diseño que espera y distribuye eventos o mensajes en un programa.

📝 Formas de manejar la asincronía en JavaScript:

• 📩 Callbacks: Una función que se pasa como argumento de otra función y que será invocada.

• 🫱🏼‍🫲🏾 Promesas: (implementado en ES6) Una promesa es una función no-bloqueante y asíncrona la cual puede retornar un valor ahora, en el futuro o nunca.

• 🛣️ Async / Await: (implementado en ES2017) Permite estructurar una función asincrónica sin bloqueo de una manera similar a una función sincrónica ordinaria.

📌 En JavaScript casi todas las operaciones de I/O (Entrada y Salida) no se bloquean. A esto se le conoce como asíncronismo. Lo único que no es procesado antes de que termine la operación son los callbacks, ya que éstos están amarrados a una operación y esperan a que sea finalizada para poder ejecutarse.

.

⏳ El asincronismo es una manera de aprovechar el tiempo y los recursos de la aplicación, ejecutando tareas y procesos mientras otros son resueltos en background (como la llegada de la información de una API), para posteriormente continuar con las tareas que requerían esa información que no tenías de manera instantánea.

⏲️ Un ejemplo fácil de asincronismo vs sincronismo es invitar a unos amigos a una fiesta y ofrecer una parrillada. Primero decides colocar la carne y verduras a la parrilla y luego repartir bebidas y algo para picar (snacks). Si fuera una persona síncrona (Blocking) tendrías que esperar a que la comida de la parrilla esté cocinada y luego atender a los invitados. Pero si fuera una persona asíncrona (Non Blocking) luego de poner la carne al carbón, sacas las bebidas frías de la nevera y compartes con los invitados mientras se cocina la carne. La acción de que la comida en la parrillada esté lista sería un callback que está esperando que finalice el proceso para ejecutarse. Pero otros procesos (como compartir la velada con bebidas y algo de picar) ya podrían irse realizando.

📥 𝗖𝗹𝗮𝘀𝗲 #𝟳: 𝗫𝗠𝗟𝗛𝗧𝗧𝗣𝗥𝗲𝗾𝘂𝗲𝘀𝘁𝟳/𝟮𝟭 📤

.

📲 XMLHttpRequest es un objeto de JS que permite hacer peticiones hacia servicios en la nube(URLs o APIs).

.

📪 Existen 5 estados en un llamado XMLHttpRequest:

.

0 → Se ha inicializado.

1 → Loading (cargando).

2 → Se ha cargado.

3 → Procesamiento si existe alguna descarga.

4 → Completado.

.

📫 Métodos y propiedades:

.

xmlhttp.open() → Prepara la petición para ser enviada tomando tres parámetros: prótocolo, url, asíncrono (true).

xmlhttp.readyState → Retorna el estado de la petición.

xmlhttp.onreadystatechange → Un eventHandler que es llamado cuando la propiedad readyState cambia.

xmlhttp.status → Retorna el estado de la respuesta de la petición. (200,400,500)

xmlhttp.send() → Envía la petición.

.

📬 Características del protocolo http:

.

Verbos: Los verbos indican acciones que están asociadas a peticiones y recursos, es decir, sirven para la manipulación de recursos cliente/servidor. Los Verbos http son:

GET → Solicita un recurso.

HEAD → Solicita un recurso pero sin retornar información, la estructura de esta petición es igual que get tanto en su headers como estatus. Es útil cuando vamos a utilizar API, para comprobar si lo que vamos a enviar esta correcto y puede ser procesado.

POST → Sirve para la creación de recursos en el servidor.

PUT → Actualiza por completo un recurso, reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

PATCH → Actualiza parcialmente un recurso.

DELETE → Elimina un recurso.

.

📭 Los códigos de estados del servidor:

.

El código de estado (status codes) sirve para describir el estado de la petición hecha al servidor.

1xx → Indican que la petición fue recibida por el servidor, pero está siendo procesada por el servidor.

2xx → Indican que la petición fue recibida, aceptada y procesada correctamente.

3xx → Indican que hay que tomar acciones adicionales para completar la solicitud.

4xx → Indican errores del lado del cliente que hizo mal una solicitud.

5xx → Indican errores del servidor. Suelen aparecer cuando existe un fallo en la ejecución en el servidor.

.

📧 Los códigos más comunes a la hora de interactuar con una API son:

.

200 → OK → Indica que todo está correcto.

201 → Created → Todo está correcto cuando se hizo una solicitud POST, el recurso se creó y se guardó correctamente.

204 → No Content → Indica que la solicitud se completó correctamente pero no devolvió información. Este es común cuando se hacen peticiones con el verbo DELETE.

400 → Bad Request → Indica que algo está mal en la petición (no encontró algo).

401 → Unauthorized → Significa que antes de hacer una solicitud al servidor nos debemos autenticar.

403 → Forbidden → Indica que no tenemos acceso a ese recurso aunque se esté autenticado.

404 → Not Found → Indica que no existe el recurso que se está intentando acceder.

500 → Internal Server Error → Indica que algo falló, es un error que retorna el servidor cuando la solicitud no pudo ser procesada.

**Códigos de estado de respuesta HTTP**

Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases:

1. Respuestas informativas (100–199),
2. Respuestas satisfactorias (200–299),
3. Redirecciones (300–399),
4. Errores de los clientes (400–499),
5. y errores de los servidores (500–599).

CallBacks Hell: Consiste en múltiples Callbacks anidados que provocan que el código se vuelva difícil de leer y ‘debuggear’ y por eso se debe evitar.

𝗖𝗹𝗮𝘀𝗲 #𝟭𝟬: 𝗤𝘂é 𝘀𝗼𝗻 𝗹𝗮𝘀 𝗽𝗿𝗼𝗺𝗲𝘀𝗮𝘀 𝟭𝟬/𝟮𝟭 🤞🏼

.

🪃 Un callback devuelve una función en los parámetros, cuando llamamos varias veces un callback, estaremos colocando muchas lineas de código y sería engorroso, por eso nacen las promesas, éstas optimizan y permiten leer mejor el código con pocas lineas.

.

🫱🏼‍🫲🏾 Las promesas son asíncronas, por lo que el código continuará su ejecución normalmente y luego dirá si la promesa se resolvió o se rechazó. Por lo que varias promesas pueden llegar a entrar en ejecución al mismo tiempo.

.

Las promesas pueden suceder:

.

Ahora

En el futuro

Nunca

.

🛠️ Para crear una promesa:

.

Utilizamos la palabra reservada new seguida de la palabra Promise que es el constructor de la promesa. Este constructor recibe un único parámetro que es una función, la cuál a su vez, recibe otros dos parámetros: resolve y reject.

El parámetro resolve se utiliza para cuando la promesa devuelve el valor correctamente.

El parámetro reject, se usa en el que caso de que no funcione.

Una Promesa puede estar en uno de los siguientes estados:

.

Pendiente pending → Una promesa inicia en este estado: no cumplida, no rechazada:

Una promesa inicialmente está pendiente.

Cumplida fulfilled → Significa que la operación se completó satisfactoriamente, .then(va => …)

Cuando llamamos a resolve entonces la promesa pasa a estar resuelta.

Cuando una promesa se resuelve entonces se ejecuta la función que pasamos al método .then

Rechazada rejected → significa que la operación falló, .catch(err => …)

Si llamamos a reject pasa a estar rechazada (obtenemos un error que nos va a indicar la razón del rechazo).

Si la promesa es rechazada entonces se ejecuta la función que pasamos a .catch

💡 𝗖𝗹𝗮𝘀𝗲 #𝟭𝟲: 𝗚𝗲𝗻𝗲𝗿𝗮𝘁𝗼𝗿𝘀 𝟭𝟲/𝟮𝟭 💡

.

Un generador en JavaScript consta de una función generadora que muestra un objeto iterable Generator. La palabra reservada yield se usa para pausar y reanudar una función generadora.

La estructura del Generador consta con la palabra function seguido de un asterísco \* : function\* ésta es una función generadora heredada.

El resultado que se quiere obtener se coloca al lado derecho de yield, puede ser de cualquier tipo (string, numérico, objetos, etc) y se puede tener tantos yield que se desee.

next() permite acceder a la función del generador y obtener con yield dos valores: value y el estado de done, es decir si tenemos yield 1; y mandamos a imprimir el resultado con next() obtenemos `{value: 1, done: false}’:

El 1 por el valor al lado derecho del primer yield.

Y done es false porque mientras haya otro yield por operar será falso.

Será true cuando se ejecute cuatro veces next() y la salida mostrará {value: undefined, done: true}. Ésto se debe a que ya no hay mas nada que mostrar, porque se mandó a imprimir un cuarto elemento y el generador solo tiene 3 yield.