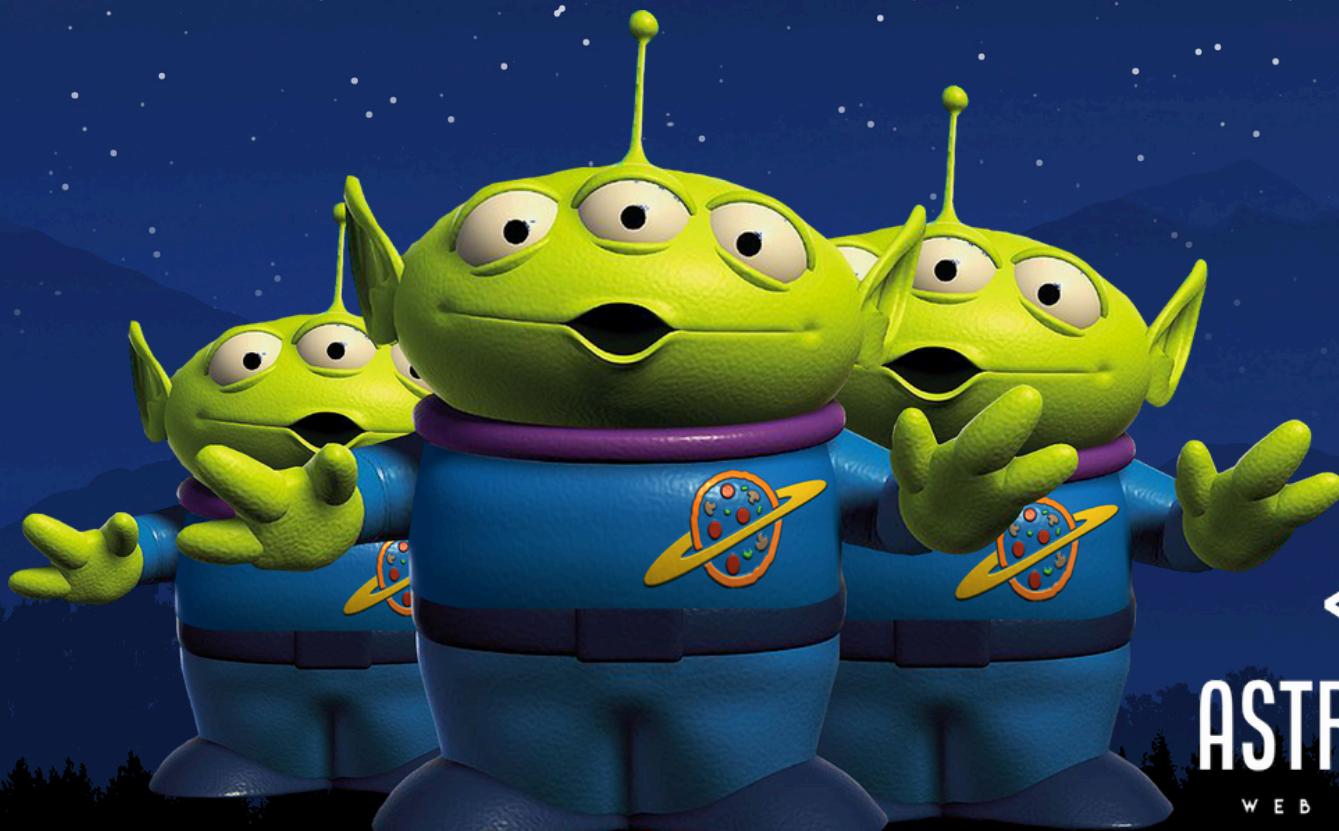


PROMESAS

EXPLICADAS CON TOY STORY

< > desde cero </ >



< ! >

ASTRID-ITZEL
WEB DEVELOPER



Parte 2: Promesas con API's en JavaScript

PROMESAS

INTRODUCCIÓN

Imagina que Woody y Buzz llaman a Pizza Planeta para hacer un pedido. La persona a cargo les dice que su orden está en camino. En ese momento, obtienen una promesa: el restaurante les promete que les entregará su comida en menos de media hora



La comida puede:

1. Llegar con **éxito** (llega y está deliciosa).
2. Llegar **con error, incompleta o no llegar** (se quema o se olvida).
3. estar todavía **en proceso** (aún se está preparando).

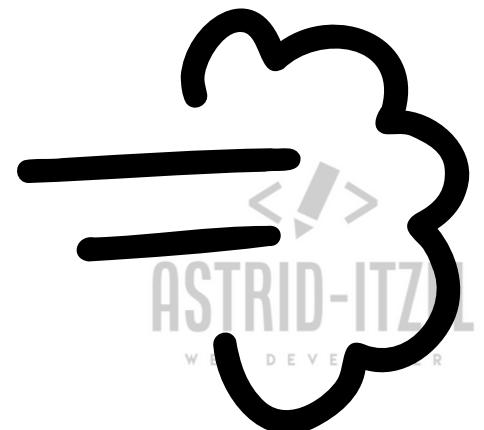
PROMESAS

¿QUÉ SON?

En JavaScript, las promesas funcionan de manera similar. Representan el eventual resultado de una operación asíncrona.

Son una forma de manejar operaciones que toman tiempo, como por ejemplo solicitudes a una API o cualquier otra cosa que no se resuelva de inmediato.

Usamos promesas para manejar los resultados cuando estén



PROMESAS

LOS 3 ESTADOS



Las promesas pueden encontrarse en uno de los tres estados posibles:

- **Cumplida (Fulfilled):**
La promesa se ha completado con éxito.
- **Rechazada (Rejected):**
La promesa ha fallado.
- **Pendiente (Pending):**
La promesa está en proceso.

PROMESAS

.THEN
.CATCH

Al trabajar con una promesa, generalmente usamos los métodos `.then()` y `.catch()` para manejar el éxito o el fracaso.

.then() Cuando la promesa se resuelve con éxito, es decir, si la pizza llegó a casa de Andy, la función dentro de `.then` se ejecutará.

.catch() Si la promesa falla, o si la pizza nunca llega, la función dentro de `.catch` se ejecutará.



PROMESAS

PROMESAS CON API

Supongamos que tenemos una API en Pizza Planeta que responde a una solicitud para pedir una pizza.

Al interactuar con API's, hacemos solicitudes HTTP como 'get' (para obtener datos) o 'post' (para enviar datos). Para hacer estas solicitudes, como obtener datos de un servidor, utilizamos la función `fetch`.

Fetch devuelve un objeto de tipo **promise**, por eso debemos encadenar un `.then()` que se ejecutará una vez que `fetch` haya terminado de obtener el recurso solicitado.

Normalmente el formato de intercambio de datos es JSON. `response.json()` toma la respuesta de la API (que viene en formato JSON) y la convierte en un objeto para que JavaScript pueda trabajar con los datos.

PROMESAS

PROMESAS CON API

Recordemos que cuando hacemos promesas, algo podría salir mal. Imagina que Sid hizo una llamada de broma a Pizza Planeta ordenando 100 pizzas y retrasó los pedidos, por ello también es necesario determinar qué pasará en caso de que la promesa falle, con **.catch()**.

Entonces, nuestro código luciría así:



```
fetch("https://api.pizzaplaneta.com/pedir") // 1. Llamar a la API
  .then((response) => response.json()) // 2. Convertir la respuesta a JSON
  .then((data) => {
    // 3. accedes a data.imagen, que debería ser la URL de la imagen de la pizza.
    console.log(data.imagen);
  })
  .catch((error) => {
    console.error("Hubo un problema:", error); // Manejar errores
  });
}
```

PROMESAS

PROMESAS CON API

Para mejorar la organización y reutilización (llamar a esta función en diferentes partes de nuestro código) podríamos encapsular el código en una función llamada pedirPizzaAPI.



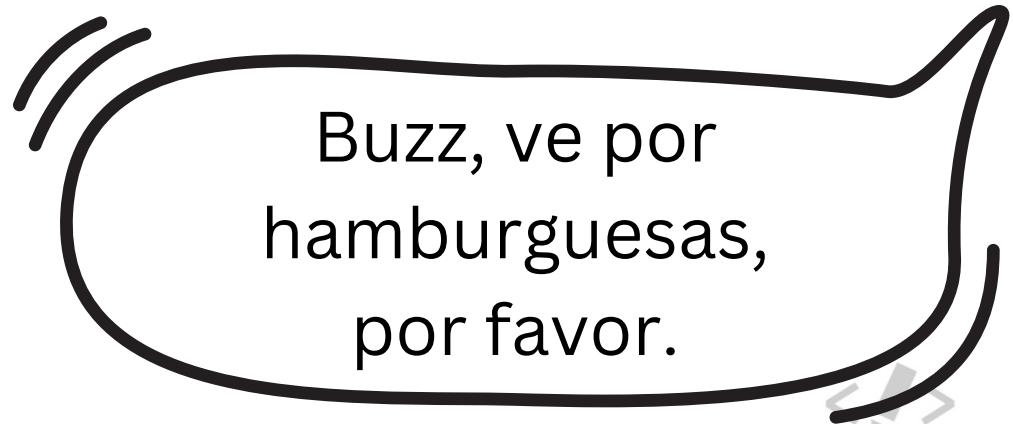
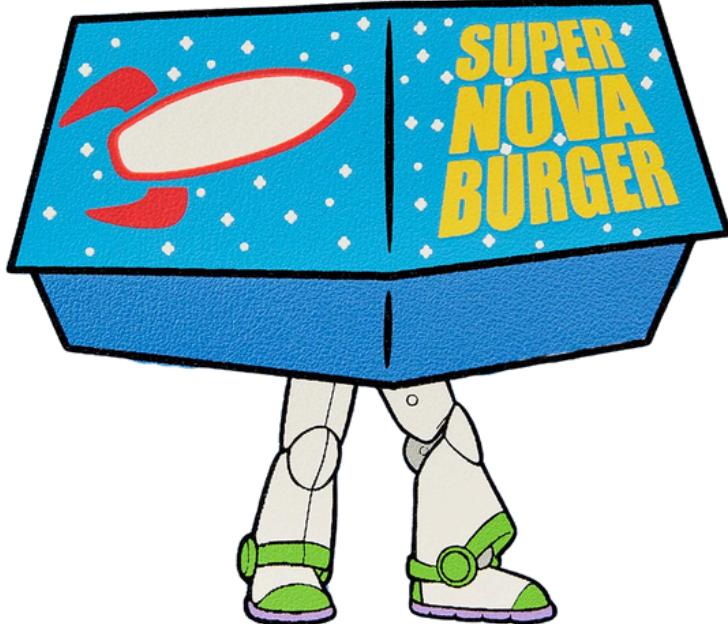
```
function pedirPizzaAPI() {
    // llamamos a una API con fetch
    return fetch("https://api.pizzaplaneta.com/pedir")
        .then((response) => {
            if (!response.ok) {
                throw new Error("Error al pedir la pizza.");
            }
            return response.json(); // Convierte la respuesta a un objeto JSON
        })
        .then((data) => {
            if (data.exito) {
                return "¡La pizza de Pizza Planeta ha llegado!";
                // Mensaje de éxito
            } else {
                throw new Error("Lo siento, la pizza se perdió en el camino.");
                // Mensaje de error
            }
        });
}
```

PROMESAS

PROMESAS CON API

Ahora, podríamos llamar y usar esta función, manejando los resultados con `.then()` y `.catch()`:

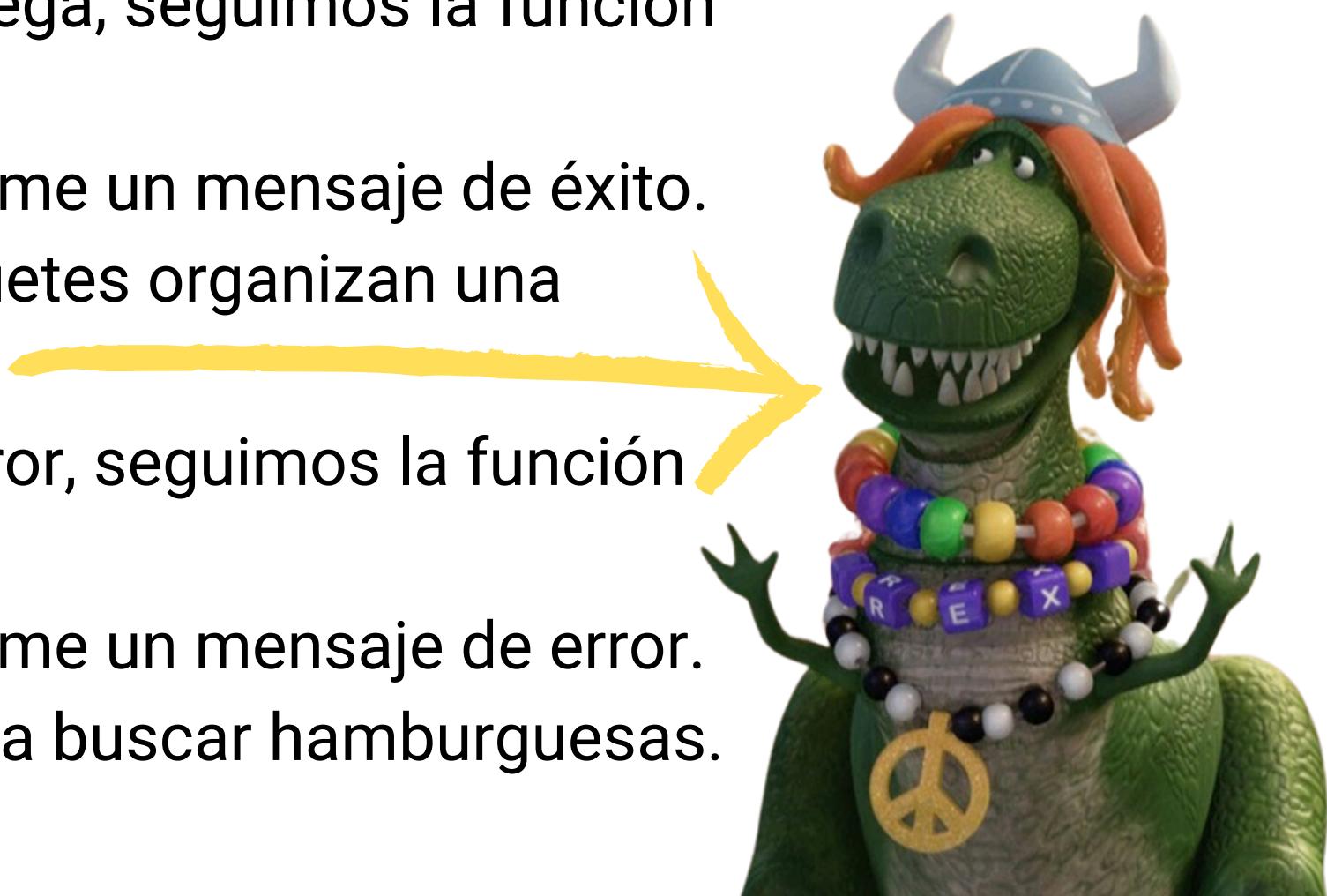
```
pedirPizzaAPI()
  .then((mensaje) => {
    console.log(mensaje); // Mensaje si la pizza llega
    console.log("¡Organizaremos una fiesta con pizza!");
  })
  .catch((error) => {
    console.log(error.message); // Mensaje si hay un error
    console.log("¡Oh, no habrá pizza hoy! Buzz, ve por hamburguesas, por favor");
});
```



PROMESAS

RESUMEN

1. Woody llama a la API de Pizza Planeta: Usa **fetch** para hacer el pedido.
2. La **API** responde: Si la pizza llega, se organiza una fiesta. Si hay un problema, Buzz tiene que ir a buscar hamburguesas.
3. Manejo del resultado:
 - Si la pizza llega, seguimos la función de **.then()**:
 - Se imprime un mensaje de éxito.
 - Los juguetes organizan una fiesta.
 - Si hay un error, seguimos la función de **.catch()**:
 - Se imprime un mensaje de error.
 - Buzz va a buscar hamburguesas.



PROMESAS

ASYNC / AWAIT



Pequeño tip extra:
utiliza async/await
para simplificar el
código

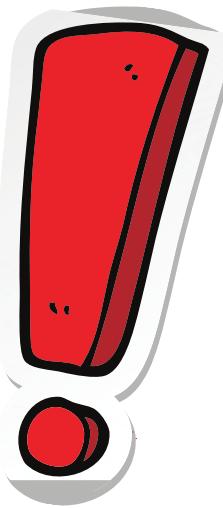
Si quieras que tu código sea más limpio y fácil de seguir, considera usar async/await. Esto facilita la comprensión del código.

async: Marca una función como asíncrona, lo que significa que puede contener operaciones que toman tiempo. Esta función siempre devolverá una promesa, incluso si no se usa explícitamente return.

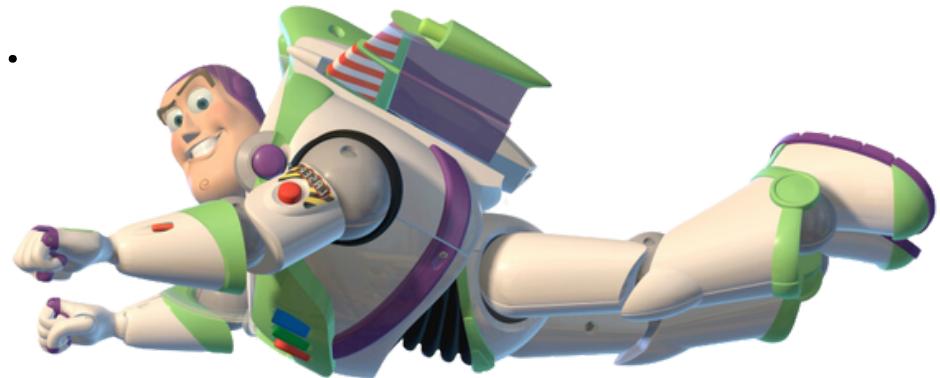
await: Se utiliza dentro de funciones asíncronas y hace que el código "espere" hasta que la promesa se resuelva, antes de continuar con la siguiente línea de código.

PROMESAS

ASYNC / AWAIT



Aunque await hace que el código "espere", solo lo hace dentro del contexto de la función asíncrona. Mientras el código de esa función está esperando, el resto del programa sigue ejecutándose. Esto significa que otras partes u operaciones de tu aplicación no se detienen.



Ventajas de async/await

- Legibilidad: el flujo del código es más fácil de seguir, ya que parece secuencial.
- Manejo de Errores: puedes usar **try/catch** para capturar errores de manera más intuitiva, lo que puede ser más natural, especialmente en bloques de código más grandes.

PROMESAS

ASYNC / AWAIT

PASO 1: Modificar la Función pedirPizzaAPI

- **Declaración de async:**

Primero, cambia la función pedirPizzaAPI para que sea asíncrona. Añadir async antes de la función permite usar await dentro de ella.

- **Uso de await:**

- await fetch(...): Hace que el código espere hasta que la llamada a la API se complete antes de continuar.
- await response.json(): Espera a que la conversión a JSON termine antes de seguir.

- **Manejo de Errores:**

Se mantiene igual, pues necesitamos lanzar un error si algo sale mal.



PROMESAS

ASYNC / AWAIT



```
async function pedirPizzaAPI() {  
    const response = await fetch("https://api.pizzaplaneta.com/pedir");  
    // Esperar a que se complete la llamada  
  
    if (!response.ok) {  
        throw new Error("Error al pedir la pizza.");  
    // Lanza un error si la respuesta no es correcta  
    }  
    const data = await response.json();  
    // Esperar a que se convierta la respuesta a JSON  
  
    if (data.exito) {  
        return "¡La pizza de Pizza Planeta ha llegado!";  
    } else {  
        throw new Error("Lo siento, la pizza se perdió en el camino.");  
        // Lanza un error si la pizza no llegó  
    }  
}
```

PROMESAS

ASYNC / AWAIT

PASO 2: Llamar a la Función con await

- Función iniciarFiesta como async: permite usar await dentro de ella.
- Uso de await al llamar la función pedirPizzaAPI: espera a que se complete la llamada a la API antes de seguir con el código.
- Manejo de Errores con try/catch: se hace más intuitivo, ya que se usa try/catch para capturar cualquier error que ocurra durante la ejecución.



```
async function iniciarFiesta() {  
  try {  
    const mensaje = await pedirPizzaAPI(); // Esperar el resultado de la función  
    console.log(mensaje); // Mensaje si la pizza llega  
    console.log("¡Organizaremos una fiesta con pizza!");  
  } catch (error) {  
    console.log(error.message); // Mensaje si hay un error  
    console.log("¡Oh, no habrá pizza hoy! Buzz, ve por hamburguesas, por favor");  
  }  
}  
  
iniciarFiesta(); // Llamar a la función
```

PROMESAS

Cuando finalmente entiendes cómo funcionan las promesas...
¡Y Buzz ya no tiene que ir por hamburguesas!



Mantente al pendiente de los próximos materiales donde explicaremos más sobre HTML, CSS, JavaScript, Git, y mucho más.



 linkedin.com/in/astrid-itzel/

 [@astriditzel.tech](https://www.instagram.com/astriditzel.tech)