PROMESAS II

Dos promesas seguidas

Deseamos lograr la bibliografía de la autora JK Rowling. Para ello, lo primero que debemos realizar es la búsqueda de la autora, y obtener su ID o Key (puede haber más de una autora con el mismo nombre) y además buscar la biografía de la autora con dicho ID/Key.

Concatenamos por lo tanto más de un *fetch()*

```
fetch('https://openlibrary.org/search/authors.json?q=j%20k%20rowling')
                .then(r => r.json())
                .then(r => {
                              console.log(r.docs[0].key)
                              return r.docs[0].key
                            })
.then(r=> fetch(`https://openlibrary.org/authors/${r}.json`))
      .then(r=>r.json())
      .then(r=>console.log(r.bio))
.catch(err => {console.error(err) } )
```

Dos promesas seguidas, async/await

Podemos realizar la misma operación pero haciendo uso del operador *await* y en consecuencia su correspondiente función *async*:

```
async function conseguir() {
     const autorID = await
fetch('https://openlibrary.org/search/authors.json?q=j%20k%20rowling')
.then(r => r.json())
.then(r => r.docs[0].key)
     console.log(autorID); // OL23919A

     const bio = await
fetch(`https://openlibrary.org/authors/${autorID}.json`)
 .then(r => r.json())
 .then(r => r.bio)


  // Returns promise to be resolved
  return {
    // this is equals to return { autorID: autorID, bio: bio}
    autorID,
    bio
  }
}


conseguir().then( r => {
             console.log(r)
             console.log( r.autorID, r.bio )
})
```

¡NOTA! Las funciones *async* siempre devuelve una promesa

Genera tu propia promesa

Supongamos que tenemos la siguiente función, y la siguiente llamada:

```
function sayHello() {
      document.write(`<h1>Kaixo</h1>`)
      // console.log("Hello!");
}
setTimeout(sayHello, 1000);
```

Pero quiero usar a modo de promesa

```
delay(1000)                        //delay no es una función ya existente.
  .then(() => sayHello())           // <-- La promesa no devuelve nada
  .catch(err => console.error(err))
```

Entonces, ¿cómo genero la promesa *delay()*?

```
// 'delay' debe devolver una nueva promesa

function delay(time)
{
    return new Promise((resolve, reject) =>

     {
           if (isNaN(time)) {

     // reject the promise. Error will be caught by .catch()
          reject(new Error('delay requires a number'));
    }
   else
    {
      // This means "after this amount of time, resolve the promise"
      setTimeout(resolve, time);
    }
  });
}
```

Usamos *delay()*, pero le indicamos una nueva función a ejecutar cuando la promesa se cumple (*resolve*):

```
function say(zer) {
      document.write(`<h1>${zer}</h1>`)
}

delay(1000) .then( r => say( r  )).catch(err => console.error(err))
```

Ahora, supongamos que el método *delay*, además de esperar, devuelve un número al azar:

```
function delay(time) {
  return new Promise((resolve, reject) => {
    if (isNaN(time)) {
      // reject the promise. Error will be caught by .catch()
      reject(new Error('delay requires a number'));
    } else {
      // This means "after this amount of time, resolve the promise"
      setTimeout(function(){ resolve( Math.random()  ) }, time);
    }
  });
}
```

CORS

Intercambio de Recursos de Origen Cruzado (CORS, de sus siglas en inglés) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un navegador/sitio web obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece. https://developer.mozilla.org/es/docs/Web/HTTP/CORS

Supongamos el siguiente ejemplo:

- Vamos al sitio web https://enable-cors.org/ y queremos coger parte de su contenido y guardarlo y visualizarlo en nuestra web. Lo ejecutamos de la siguiente manera y nos percatamos de que NO HAY PROBLEMA.

```
fetch("https://enable-cors.org/")
  .then(response => response.text())
  .then(text => {
    const parser = new DOMParser();
    const htmlDocument = parser.parseFromString(text, "text/html");
    const section =
htmlDocument.documentElement.querySelector("section");
    document.write(section.innerHTML)
  })
```

¡NOTA! DOMParser

En el código visualizamos como para obtener el contenido dentro del del *fetch* hace uso de DOMParser().

DOMParser, analiza gramaticalmente (parsear) código XML o HTML almacenado en una cadena de texto y convertirlo en un documento DOM (https://developer.mozilla.org/es/docs/Web/API/DOMParser).

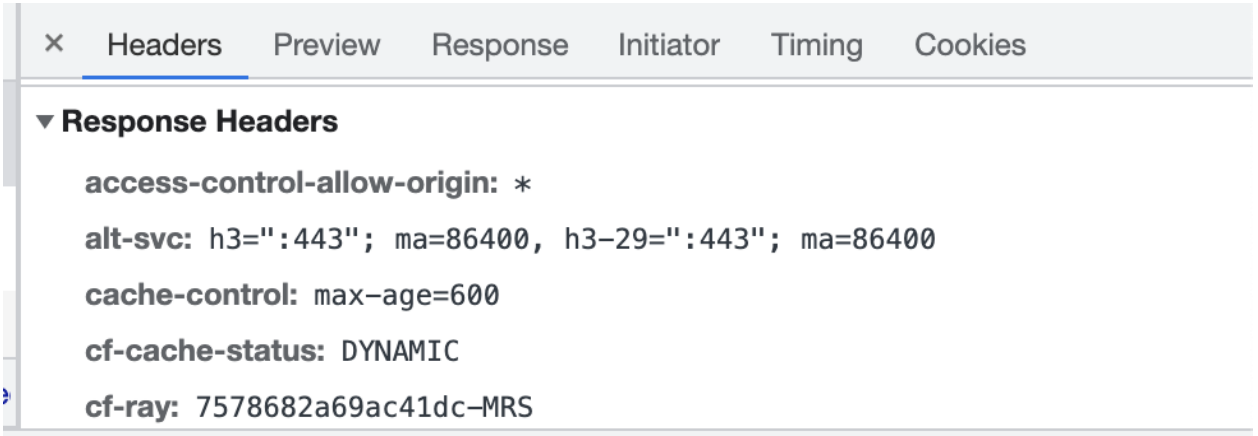
parseFromString -> https://developer.mozilla.org/en-US/docs/Web/API/DOMParser/parseFromString
¡**NOTA! document.querySelector()**
querySelector() -> https://developer.mozilla.org/es/docs/Web/API/Document/querySelector

– Pero, ¿si quisiéramos hacer lo mismo, para obtener el precio de un libro en concreto y visualizarlo en nuestra web, adquiriendo dicho dato desde la web de Amazon? EXISTE UN PROBLEMA

```
fetch("https://www.amazon.com/dp/1491920491")
  .then(response => response.text())
  .then(text => {
    const parser = new DOMParser();
    const htmlDocument = parser.parseFromString(text, "text/html");
    const section =
htmlDocument.documentElement.querySelector("#newBuyBoxPrice");
    document.write(section.innerText)
  })
```

En el caso de amazon, nos ocurre que no tenemos permitido acceder a dichos recursos desde un dominio diferente. ¿Cómo lo verificamos?

Enabled
https://openlibrary.org/api/books?bibkeys=ISBN:0451526538&format=json
enable-cors.org
ikasten.io:3000



Disabled

https://www.amazon.com/dp/1491920491

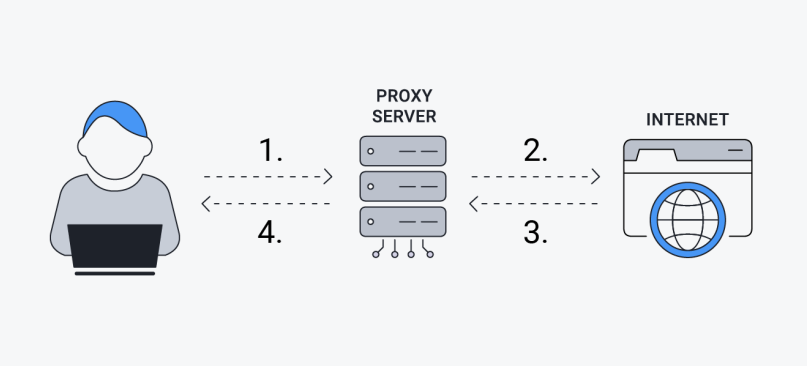
SOLUCIÓN

https://github.com/Rob--W/cors-anywhere/
CORS Anywhere is a NodeJS reverse proxy which adds CORS headers to the proxied request.

Es decir, esta solución nos permite realizar la consulta de arriba,, evitando el problema de CORS haciendo uso de un proxy para ello.

¡NOTA! Proxy

¿Qué es un Proxy?
Un proxy es un equipo informático que hace de **intermediario** entre las **conexiones de un cliente y un servidor de destino**, filtrando todos los paquetes entre ambos. Siendo tú el cliente, esto quiere decir que **el proxy** recibe tus peticiones de acceder a una u otra página, y **se encarga de transmitir al servidor de la web para que esta no sepa que lo estás haciendo tú.**



[1]

Así, accedo al sitio web de amazon deseado mediante un proxy(en este caso https://ikasten.io:3000)

http://ikasten.io:3000/https://www.amazon.com/dp/1491920491

Si ahora realizo la misma consulta, con esta nueva dirección, NO HAY PROBLEMA

```
fetch("http://ikasten.io:3000/https://www.amazon.com/dp/1491920491")
  .then(response => response.text())
  .then(text => {
    const parser = new DOMParser();
    const htmlDocument = parser.parseFromString(text, "text/html");
    const section =
htmlDocument.documentElement.querySelector("#newBuyBoxPrice");
    document.write(section.innerText)
  })
```

[1] Fuente: https://www.avg.com/es/signal/proxy-server-definition

MÁS DE UNA PROMESA AL MISMO TIEMPO

Vamos a tener en cuenta las consultas y ejemplos realizados hasta el momento. Supongamos que queremos ejecutar **más de una promesa/consulta**, pero **al mismo tiempo**. Es decir, no queremos esperar a que acabe una promesa para empezar a ejecutar la siguiente.

En el ejemplo de Jk Rowling y la biografía, ocurría lo contrario. Primero obtenemos el *key* de la autora, y luego con ese *key* la biografía. Es decir, era necesario que la primera promesa se ejecutase en su totalidad para seguir con la siguiente.

En este caso,

- Con la primera promesa queremos obtener el título del libro:

https://openlibrary.org/api/books?bibkeys=ISBN:1491920491&format=json&jscmd=details

(JSON path --> ."ISBN:1491920491".details.full_title)

- Con la segunda en cambio, el precio de dicho libro:

https://www.amazon.com/dp/1491920491

¡NOTA! promise.all

Promise.all

Hacemos uso del método que nos ofrece JavaScript (promise.all), dónde se devuelve una promesa que termina correctamente cuando todas las promesas en el argumento han sido concluidas con éxito.
(https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise/all)

SOLUCIÓN
https://gist.github.com/juananpe/ed1830f4a2eb785585252b61da4e3fb9

CARGANDO IMÁGENES

(about: blank)

```
let irudi = new Image()
irudi.src =
"https://www.berria.eus/argazkiak_jarraia/egunekoak/2022-10-09
/jarraia219276.jpg?format=webp&width=360&quality=75"
irudi.width
irudi.heigh
```

EJERCICIOS

1- Mapa /JSON
https://github.com/juananpe/mapa-json

2- Implementa *loadJSON* como una promesa

Tenemos que poder realizar la siguiente instrucción

```
loadJSON(url).then(r => console.log( r ) ).catch( e => console.log(e) )
```

3- Implementa *loadImage* a modo de promesa

```
loadImage(url)
          .then(img => console.log(`w: ${img.width} | h:
${img.height}`))
        .catch(err => console.error(err));
```