Técnicas de Programación

Carrera Programador full-stack

Promesas

Conversemos con ES7

Ejemplo de cómo bajar algo del servidor

Usuario: - Hola ES7, mucho gusto. Me dijeron que con vos puedo bajar cosas de un servidor más fácil

ES7: - Sí! Es mucho más fácil, solo tenes que usar el método fetch(URL)

Usuario: - Genial!

Conversemos con ES7

Usuario: - fetch(http://...), ahora me das el archivo?

ES7 :- No, no, tarda mucho eso, pero te prometo que lo bajo

Usuario: - Pero lo necesito para seguir

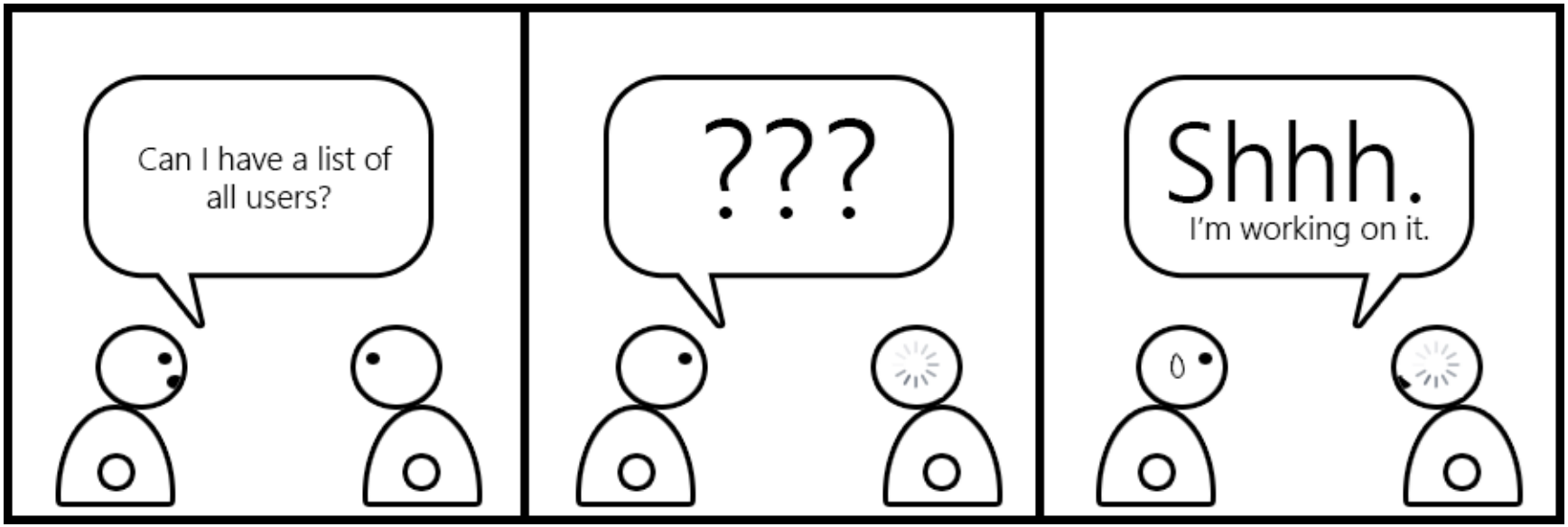
ES7: - Ya te dije que te lo prometo! Vos decime que queres que haga después con el archivo

*Versiones de ES6/7/8/9

https://medium.com/@madasamy/javascript-brief-history-and-ecmascript-es6-es7-es8-features-673973394df4

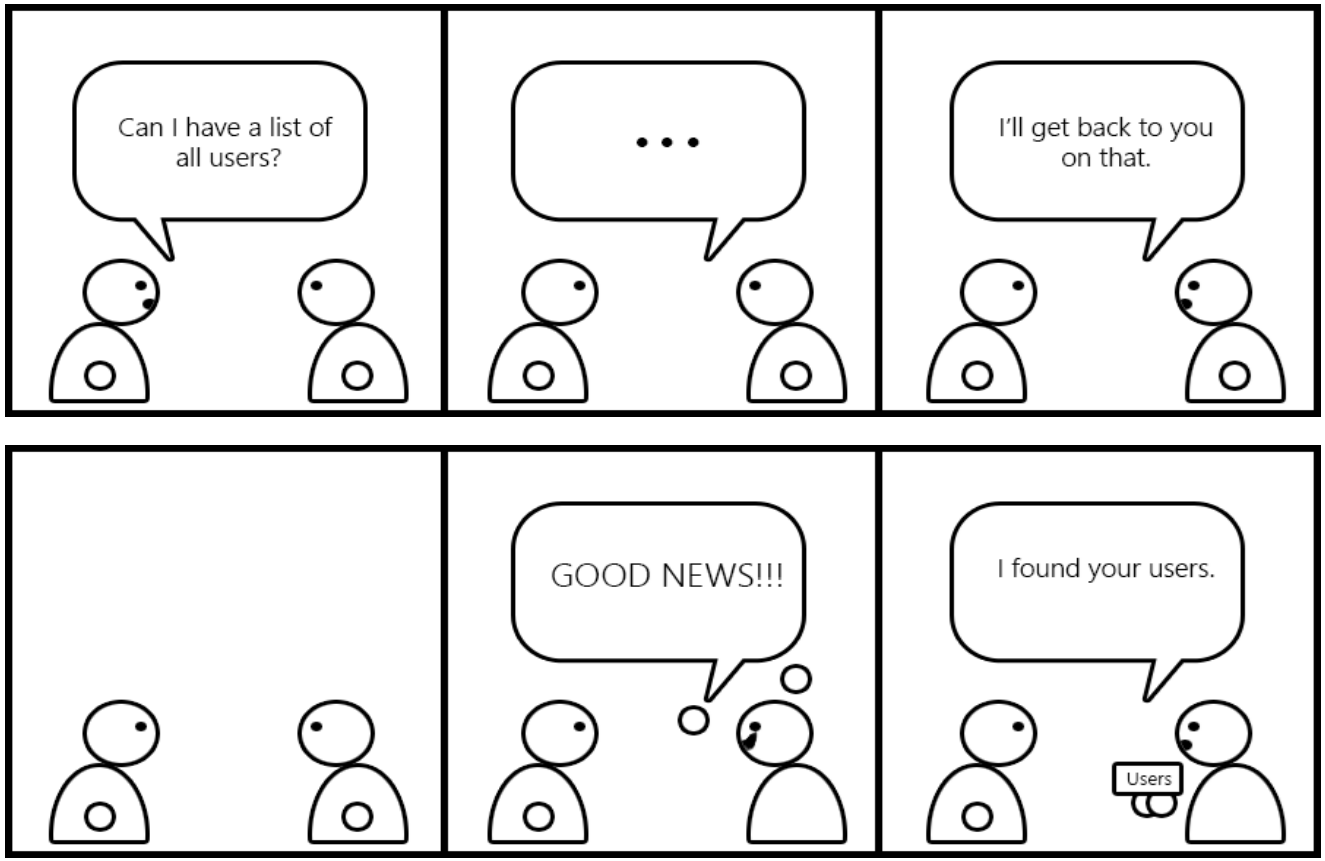
Llamados asincrónicos… ¿Por qué usarlos?

JavaScript es de un solo hilo, es decir, dos porciones de secuencia de comandos no se pueden ejecutar al mismo tiempo, tienen que ejecutarse uno después del otro. Sin llamados asíncronos, quedamos esperando una respuesta.



Promises

ES6 introduce Promises (promesas en castellano). Son un objeto que representa la terminación o el fracaso eventual de una operación asíncrona

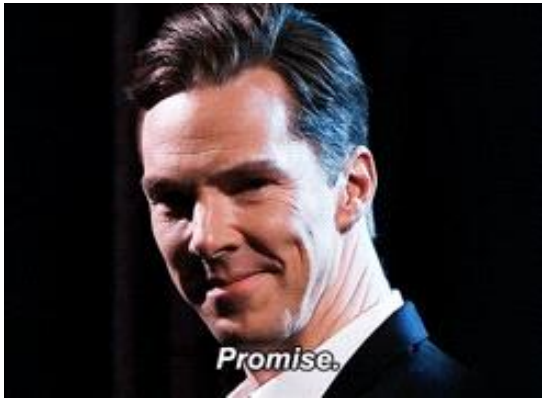


https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Promises

Las promesas son una característica más reciente que fue añadida en la versión 6 del estándar ECMAScript. Una Promise maneja un evento único cuando una operación asíncrona se completa o falla.

Una promesa se define como una función no-bloqueante y asíncrona cuyo valor de retorno puede estar disponible justo en el momento, en el futuro o nunca.



Promises

ES7 incorpora la interfaz **fetch()**

```
let promise = fetch(url);
promise.then(response => …do something… ).catch()
```

O la versión corta

```
fetch(url).then(response => …do something… ).catch()
```

El uso más simple de fetch() toma un argumento (la ruta del recurso que se quiera traer) y **el resultado es una promesa** que contiene la respuesta (un objeto Response)

https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch

Promises

¿Qué está ocurriendo en cada llamado a la función **then()**?

```
fetch('/file.html')
  .then(function(r){
      return r.text()
  })
  .then(function(html) {
     console.log(html);
  })
  .catch(function(e) {
    console.log("Booo");
  })
```

Respuesta de la solicitud fetch

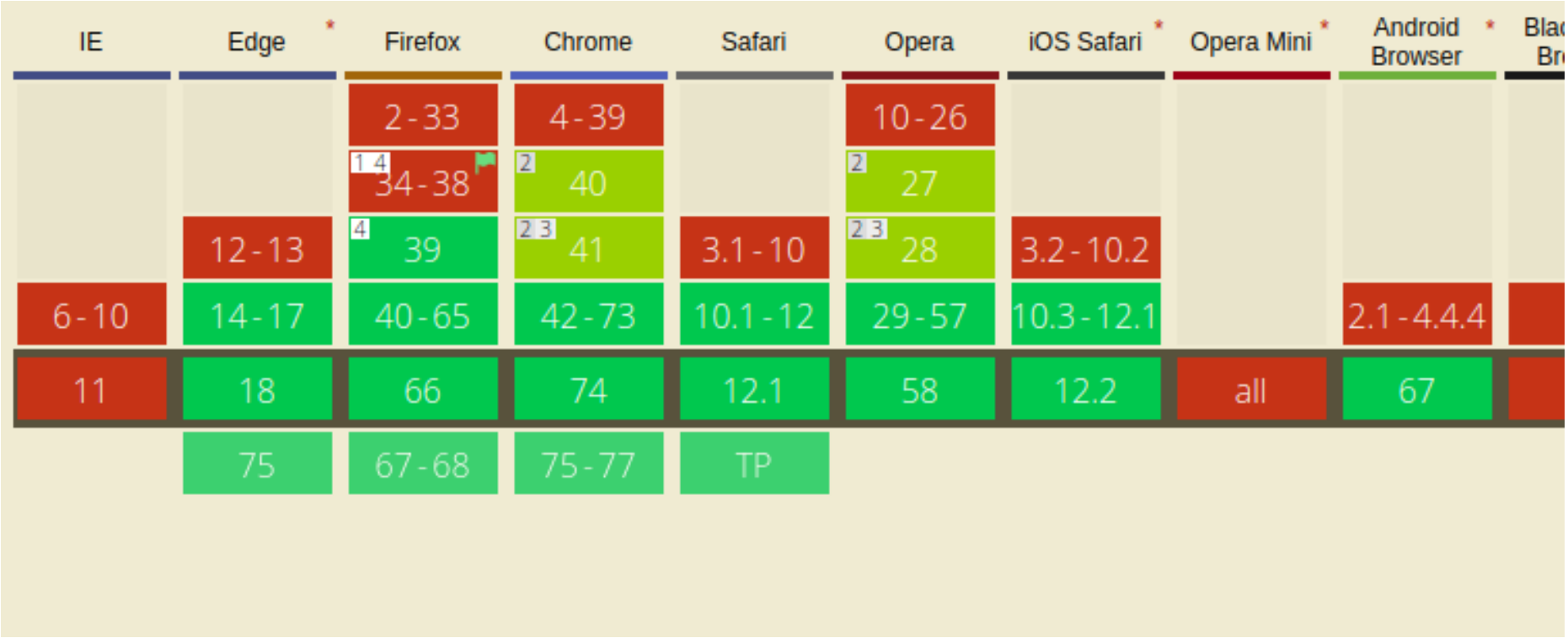
Procesamiento de la respuesta (Nos da otra promesa)

Respuesta procesada

Error de conexión

Soportes en Navegadores

Fetch funciona en los navegadores modernos más populares.



Creando una Promise

Para crear un objeto "Promise" voy a tener que entregarle una función, la encargada de realizar ese procesamiento que va a tardar algo de tiempo. En esa función debo ser capaz de procesar casos de éxito y fracaso y para ello recibe como parámetros dos funciones:

La función "**resolve**": la ejecutamos cuando queremos finalizar la promesa con éxito.
La función "**reject**": la ejecutamos cuando queremos finalizar una promesa informando de un caso de fracaso.

Código de una Promise

```
// Esta funcion devuelve una promesa
function hacerAlgoPromesa() {
  return new Promise( function(resolve, reject){
    console.log('hacer algo que ocupa un tiempo...');
    setTimeout(resolve, 1000);
  })
}
```

Como se puede ver, nuestra función **hacerAlgoPromesa()** devolverá siempre una promesa (return new Promise), entonces se debe **“suscribirse”** y esperar con **then** y **catch.**

Código de una Promise

```
// Se espera resultado de la promesa
hacerAlgoPromesa()
  .then( function() {
    console.log('la promesa terminó.');
  }),
 .catch(function(err) {
    console.log('hemos detectado un error', err');
  });
```

Esta subscripción a la promesa la tomamos con **then()** para el caso satisfactorio y **catch()**, en caso de que haya fallado.

Response de una Promise

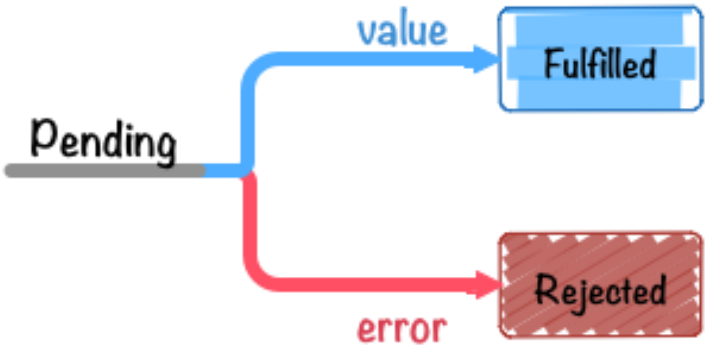
El objeto “response” tiene información de la respuesta obtenida del servidor.

Con response .ok nos dice si la descarga pudo hacerse correctamente (Código HTTP 200).

Terminología de una Promise

Una promesa tiene 4 estados

- Cumplida *(fulfilled)*
- Rechazada *(rejected)*
- Pendiente *(pending)*
- Finalizada *(settled)*



Término **then**

En vez de pasar funciones callback a una función, a la promesa le “encadenamos” las funciones callback.

```
hazAlgo(exitoCallback, falloCallback);
```

Promise

```
hazAlgo().then(exitoCallback, falloCallback);
```