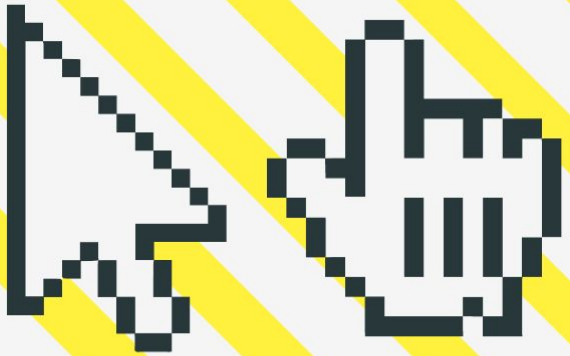
Curso de

Asincronismo con JavaScript

Oscar Barajas | @gndx



Qué es el asincronismo

Los lenguajes de programación son sincrónicos...

JavaScript es síncrono por defecto y tiene un solo subproceso.

JavaScript es síncrono y no bloqueante, con un bucle de eventos (concurrencia), implementado con un único hilo para sus interfaces de I/O.

JavaScript es single-threaded

Aún con múltiples procesadores, solo se pueden ejecutar tareas en un solo hilo, llamado el hilo principal.

Bloqueante:

Una tarea no devuelve el control hasta que se ha completado.

No bloqueante:

Una tarea 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.

Concurrencia en JavaScript

Utiliza un modelo de concurrencia basado en un "loop de eventos".

EventLoop

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... (ES6)

Función no-bloqueante y asíncrona la cual puede retornar un valor ahora, en el futuro o nunca.

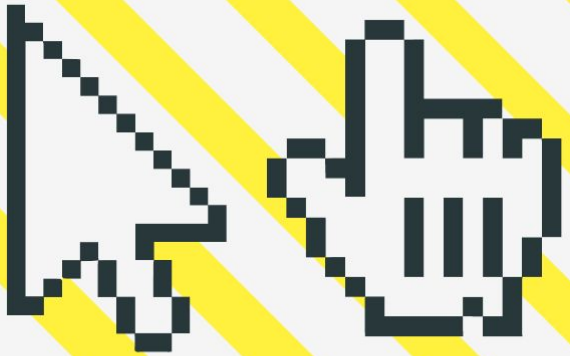
Async / Await... (ES2017)

Permite estructurar una función asincrónica sin bloqueo de una manera similar a una función sincrónica ordinaria.

JavaScript acaba de convertirse en Multi-Threaded con la capacidad de realizar múltiples tareas simultáneamente.

Esto cambia nuestra definición de JavaScript...

JavaScript es: asíncrono y no bloqueante, con un bucle de eventos (concurrencia) implementado con un único hilo para sus interfaces de I/O.



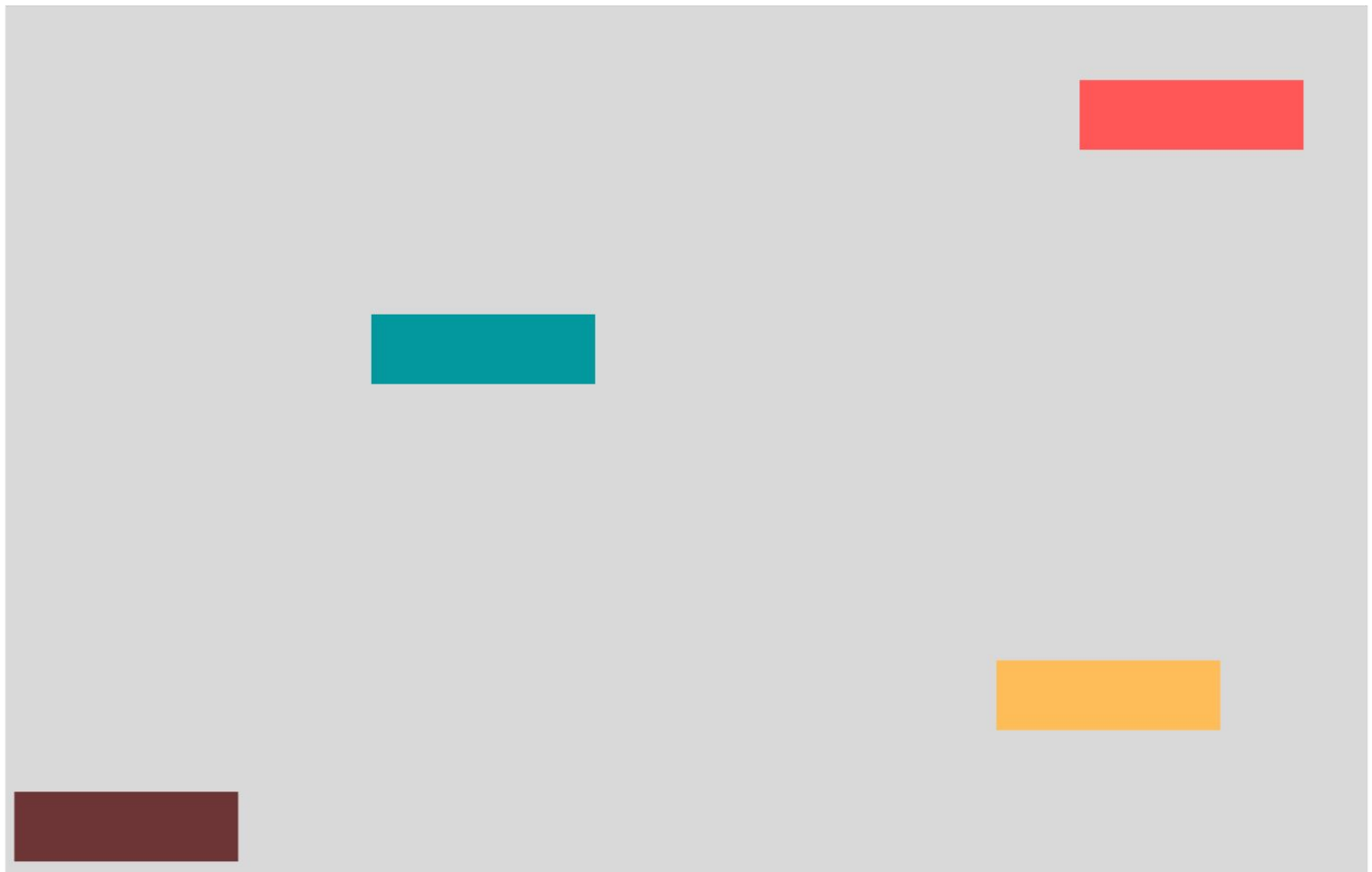
Event Loop

Event Loop

El bucle de eventos es un patrón de diseño que espera y distribuye eventos o mensajes en un programa.

Memory Heap

Los objetos son asignados a un montículo (espacio grande en memoria no organizado).

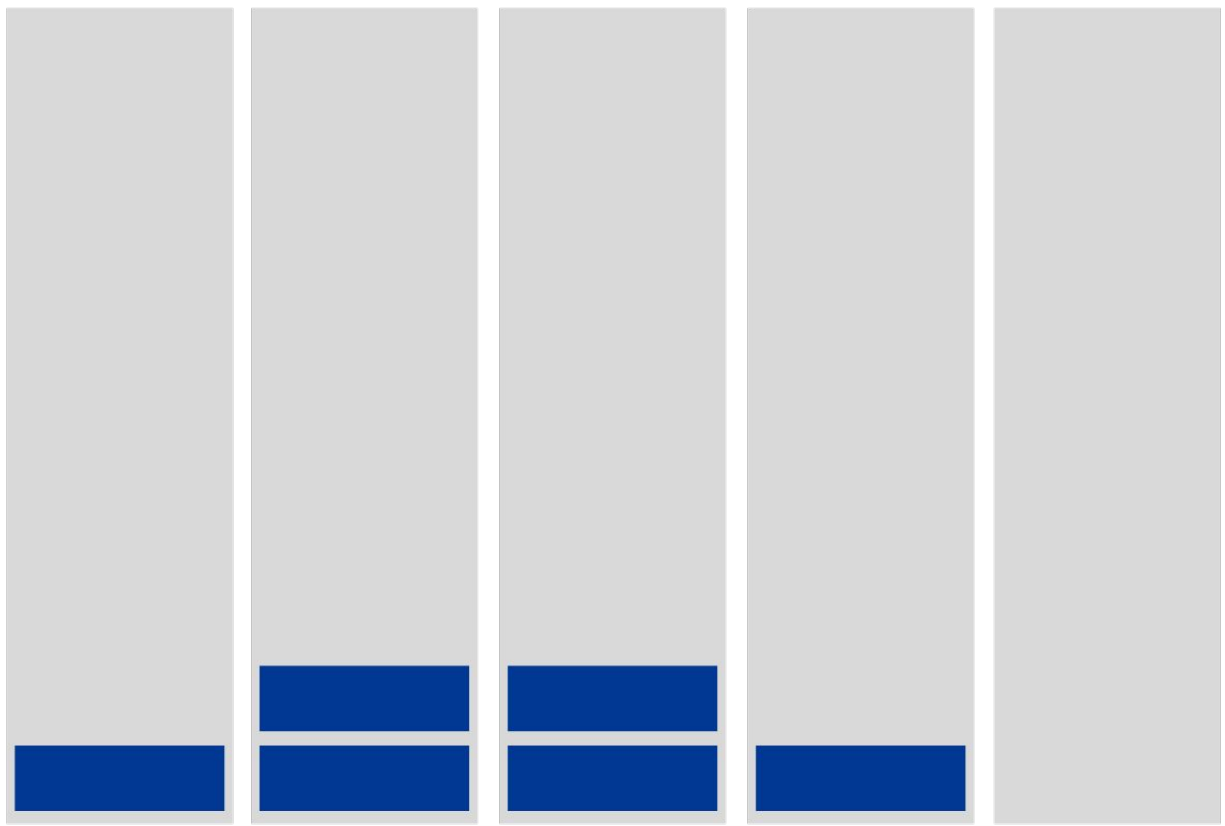


Call Stack (pila)

Apila de forma organizada las instrucciones de nuestro programa.

Call Stack

LIFO (Last-in, First-out)



push pop

TASK QUEUE

Cola de tareas, se maneja la concurrencia, se agregan las tareas que ya están listas para pasar al Stack (pila).

El stack debe de estar vacío.

Task Queue

LIFO (Last-in, First-out)

MicroTask Queue

Las promesas tienen otra forma de ejecutarse y una prioridad superior.

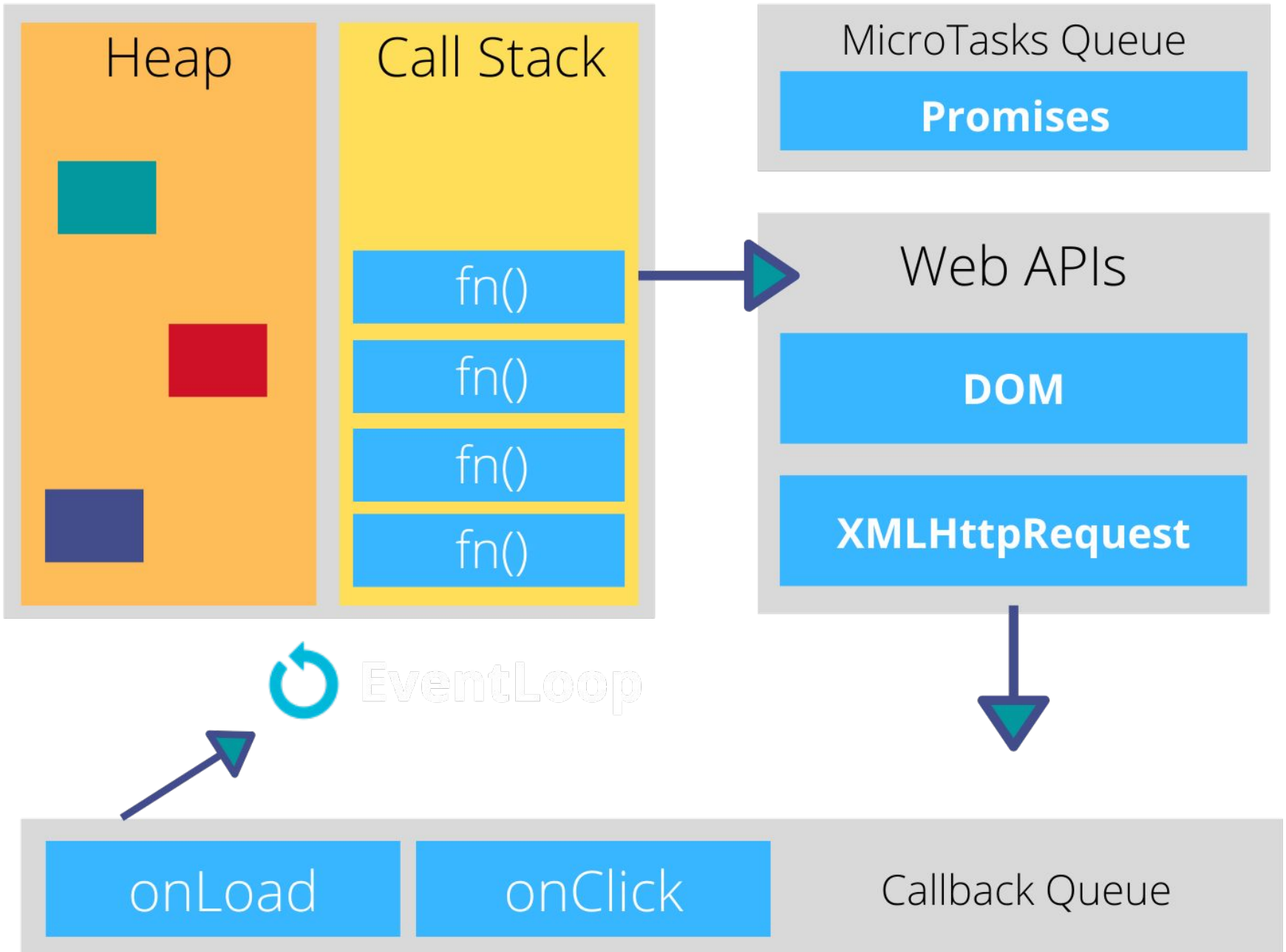
Web APIs

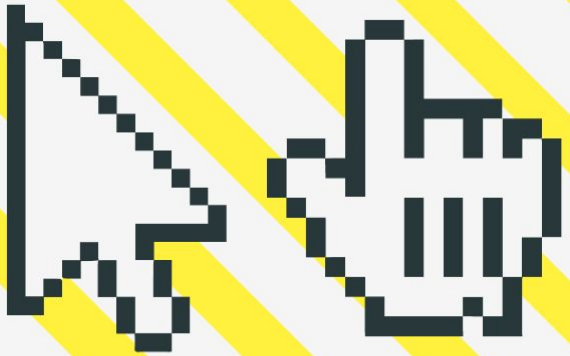
JavaScript del lado del cliente: setTimeout, XMLHttpRequest, File Reader, DOM.

Node: fs, https.

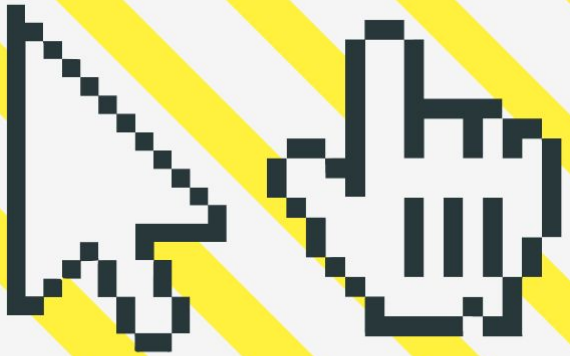
Event Loop

Tarea asignada para mover del Task Queue al Stack, solo si el Stack está vacío.

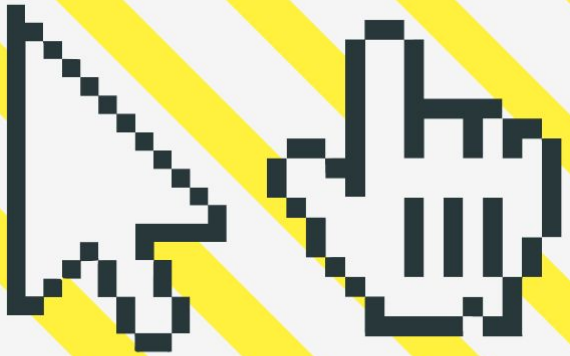

Heap
Call Stack
fn()
fn()
fn()
fn()
MicroTasks Queue
Promises
Web APIs
DOM
XMLHttpRequest
EventLoop
onLoad
onClick
Callback Queue



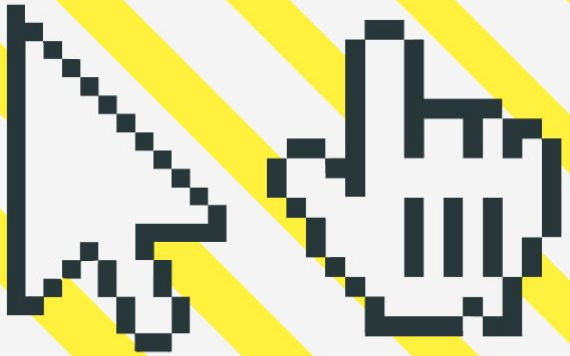
Configuración



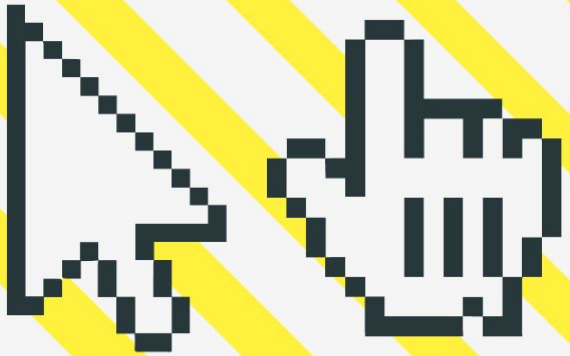
Callbacks



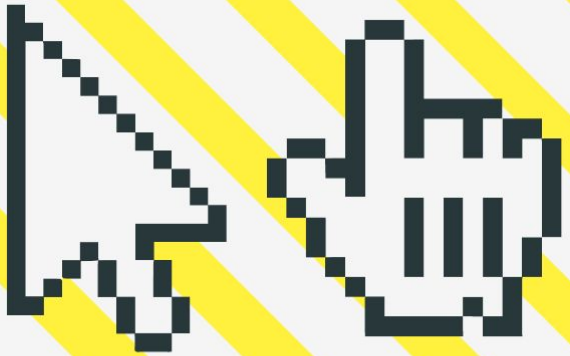
XMLHTTP Request



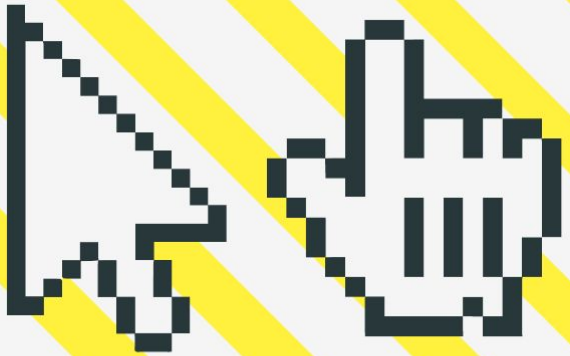
Fetch data



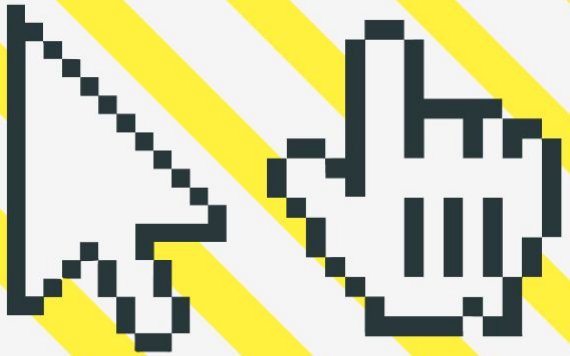
Promise



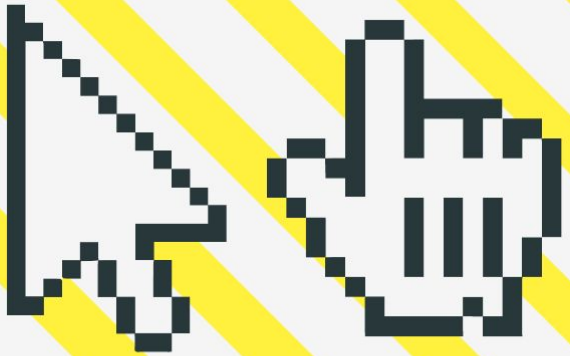
Fetch



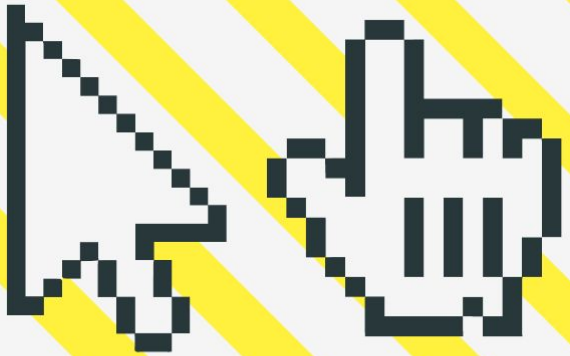
Fetch POST



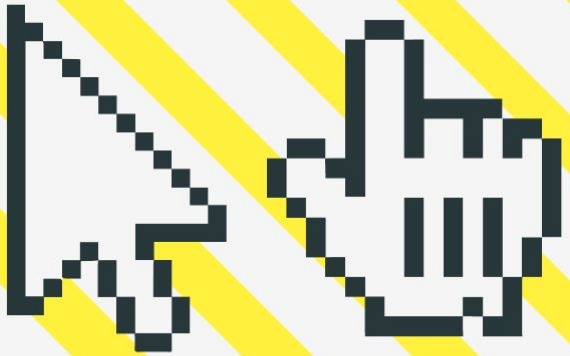
Funciones asíncronas



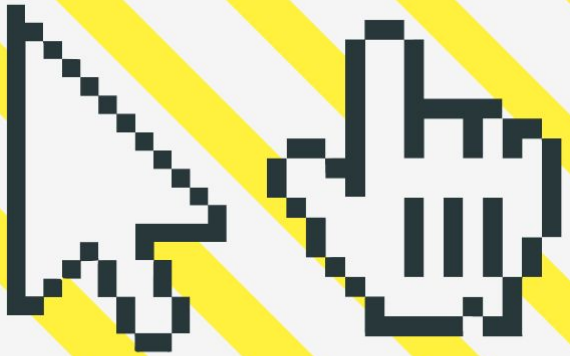
Try catch



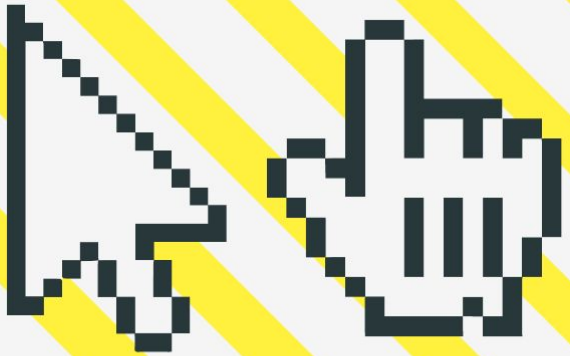
Project



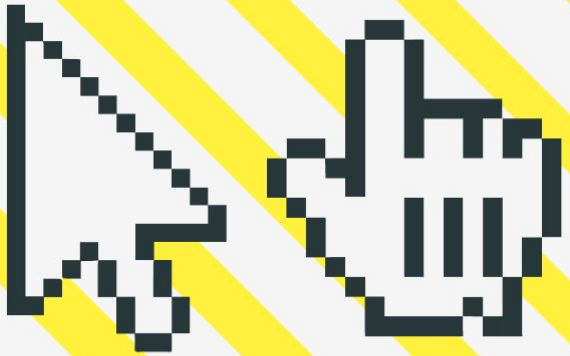
Consumiendo APIs



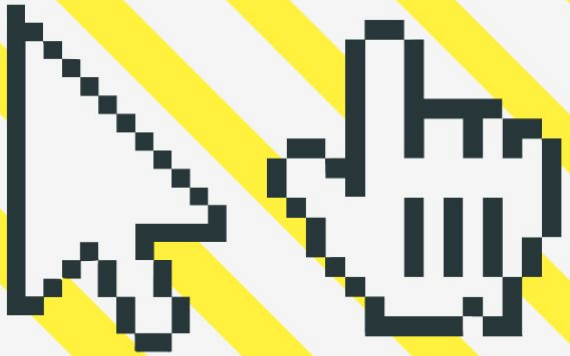
Deploy



Generators



Próximos pasos



Próximos Pasos...