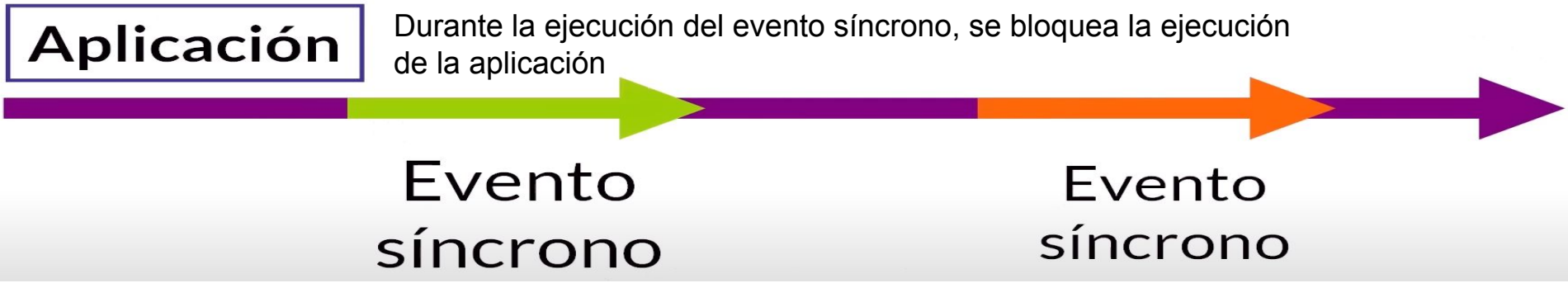
Programación Web 2 Servidor Eventos y Asincronía

U-TAD

Eventos en Node.js



Eventos en Node.js

Evento: Una acción que se realiza en la aplicación (por ejemplo, leer o agregar datos a una Base de Datos).

El servidor está escuchando por si hay alguna petición por parte del cliente. El evento sería la acción que desencadena un proceso.

También podemos definir eventos que se desencadenen y ejecuten internamente en el servidor (por ejemplo, con *emmiters*).

Emitters (emisores): Objetos que emiten eventos nombrados y llaman a funciones específicas cuando ocurren.

- Son instancias de la clase **EventEmitter**.
- Tienen un método .**on()** para asociar una función al evento.
- Esa función se ejecuta cuando ocurre el evento. Y se denomina **“Event Handler”** (o “event listener”).

Módulo events

Nos permite: definir, emitir y escuchar.

Para ello, en index.js:

```
const EventEmmitter = require('events'); //devuelve una clase por eso la "E"

const emisorProductos = new EventEmitter();

emisorProductos.on('eventName', () => { … });

//Podemos emitir el evento donde lo necesitemos con

emisorProductos.emit('eventName');
```

El programa busca la función asociada al evento para ejecutarla. Si no se lanza el .emit no se ejecuta la función asociada al evento.

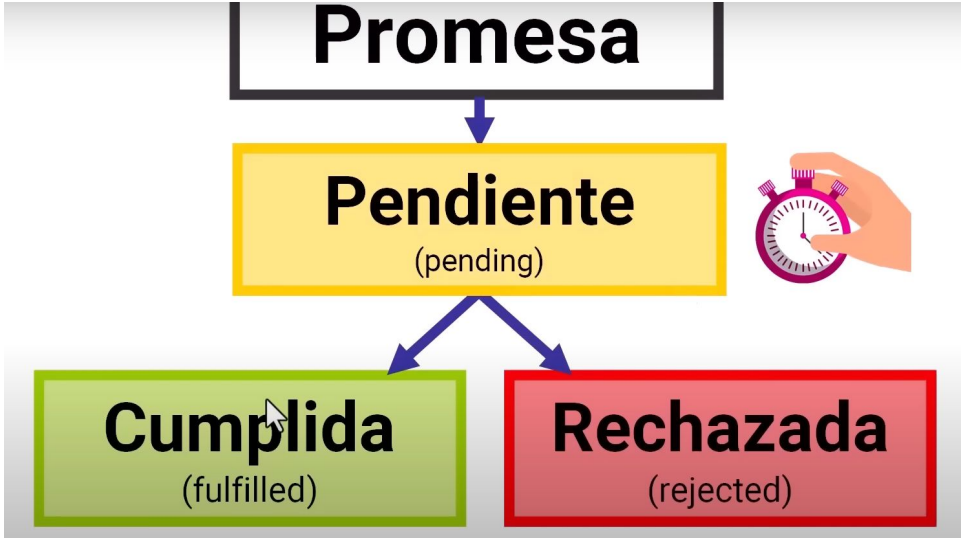
También podemos pasarle parámetros a esas funciones **events handlers**:

```
(param1, param2, …) => { … };  .emit('eventName', param1, param2, …);
```

Promesas en JavaScript

Promise: Objeto de JavaScript que representa el eventual resultado (o error) de una operación asíncrona (no conocemos el resultado, o el error, hasta que se complete el proceso asíncrono).

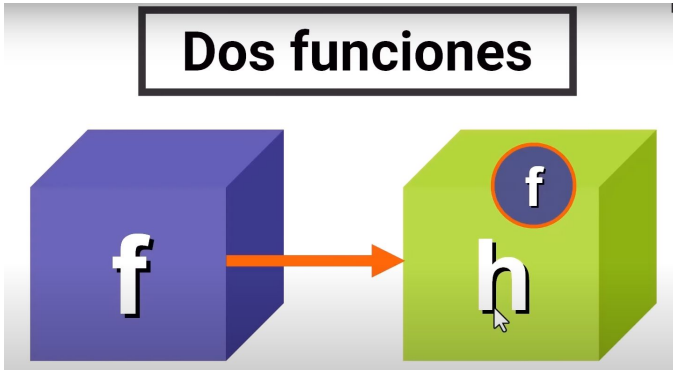
Cuando se inicia la promesa se inicializa en estado “pending”. Después de un indefinido tiempo, podrá estar en la fase “fulfilled” o “rejected”.



Promesas en JavaScript

El Objeto de la Promesa se asocia a una **callback function** (se ejecuta cuando se completa el proceso asíncrono para procesar el resultado).

Función callback: Función que se pasa a otra función como argumento y luego se ejecuta dentro de la función externa.



Las promesas tienen un método **.then()**, con el cual podemos decidir qué ocurre cuando se completa la promesa (éxito o error).

Promesas

- Sintaxis

```
const miPromesa = new Promise((resolve, reject) => { … });
const manejarPromesaCumplida = (valor) => { … };
const manejarPromesaRechazada = (razonRechazo) => { … };
miPromesa.then(manejarPromesaCumplida, manejarPromesaRechazada);
//o bien: miPromesa.then(manejaPromesa).catch(manejarRechazo);
```

- Sintaxis alternativa en **.then** (Method Chaining):

```
const miPromesa = new Promise((resolve, reject) => { … });
miPromesa
    .then((valor) => { … });
    .catch((razonRechazo) => { … });
```

Ejercicio Promesas

Tenemos una tienda de pizzas. El sistema de pedidos realiza acciones asíncronas, puede tardar X segundos. Y digamos que como el 20% de las veces puede fallar por problemas de red o de saturación.

- Simular una función con el 80% de éxito.
- Crear una Promesa simulando la asincronía con setTimeout de 3000ms.
- En el condicional del setTimeout usa el valor devuelto por la función de simulación de éxito.
- Define las funciones para manejar si fue bien o mal.
- Indica por consola si el proceso fue exitoso o falló, asociando las funciones anteriores a la Promesa con el método .then()

Encadenar promesas:

```
functionAsync1()
    .then(response1 => {
      …
      return functionAsync2()
    })
    .then(response2 => {...})
    .catch(err => {...});
```

Promesas con async / await (uso actual)

```
async functionAsync {
    try {
      … //Si queremos esperar a que finalice la Promesa, usamos await
      const response1 = await functionAsync1();
      …
      const response2 = await functionAsync2();
    } catch (err) { … }
}
```

Ejercicio

1- Crea un array de objetos de productos tal que:

```
let products = [
    {
        nombre: "PC-Gaming",
        marca: "Asus",
        precio: 1200
    },
    {
        nombre: "Tablet",
        marca: "Samsung",
        precio: 300
    },
    {
        nombre: "Cámara-Reflex",
        marca: "Canon",
        precio: 650
    }
]
```

2.- Crea una función getProducts() que devuelva una Promesa con esos productos (simula la asincronía con setTimeout de 3000 ms).

3.- Llama a getProducts() con .then() para obtener los productos.

4.- Llama a getProducts() con await dentro de una función async para obtener los productos.