NodeJS avanzado: Callback y promesas

Gabriel Rodríguez Flores

October 14, 2021

ADA ITS
INSTITUTO TECNOLÓGICO SUPERIOR

    - Callback
- Promises
- Async/Await

Contents

1 Teoría

1.1 Event loop

- Pila de ejecución

1.2 Callbacks

- Callbacks y asincronia
- Video callbacks

1.2.1 Callback hell

- callback hell
- callback hell 2
- callback hell historia
- Callback hell web

1.3 Promesas

- Teoria
- Sintaxis
- Uso promesas
- Tutorial Promesas

1.3.1 Resolver promesas

- Clase Promise
- Then
- Promise.resolve / Promise.all
- await

1.4 Sync vs Async

- async/await
- async/await 2

- En librerías (ejemplo FileSystem)
- En la programación (solo cuando sea necesario)

1.5 Buenas prácticas

- No resolver promesas en bucles. Crear array de promesas y resolver al final.
- Buenas practicas 1
- Buenas practicas 2
- Buenas practicas 3

2 Ejemplos

- Situación actual y problemática

```
console.log('Espera')
setTimeout(() => console.log('...'), 1000);
console.log('Ya!')
```

- Presentación de promesas

```
const promise = new Promise(function(resolve, reject) {
  resolve('Success!');
  // or
  // reject ("Error!");
});

promise.then(function(value) {
  console.log(value); // Success!
}, function(reason) {
  console.log(reason); // Error!
});
```

- Simplificación de escritura de promesas

```
const promise = new Promise((resolve, reject) => {
  resolve('Success!');
  // or
  // reject ("Error!");
});

promise
  .then((value) => console.log(value))
  .catch((reason) => console.log(reason));
```

- Tiempo de espera usando promesas

```
console.log('1');

const promise = new Promise(
  function (resolve, reject) {
    // console.log('!!');
    setTimeout(() => {
      console.log('resolve');
      resolve('resolve');
    }, 1000);
  }
);

console.log('2');

promise.then(
  function(val){
    console.log('done')
  }
);

console.log('3');
```

- Uso de async/await para programación funcional

```
/* Async / Await */
function sleep(ms) {
  return new Promise(resolve => {
    setTimeout(() => {
      console.log('resolve');
      resolve('resolve');
    }, ms);
  });
}

function init(){
  console.log('1');
  sleep(1000);
  console.log('2');
}

/*async function init(){
  console.log('1');
  await sleep(1000);
  console.log('2');
}*/

init();
console.log('3');
```

- Promesas dentro de los bucles (problemática)

```
function timeout(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      return resolve(x);
    }, x);
  });
}

async function init(){

const promises = [];
const results = [];
//First loop, array creation
for (let i = 0; i < 20; i++) {
  const promise = await timeout(i * 100).then(x => results.push({
    index: i,
    timeout: x
  }));
  promises.push(promise);
}
/*Promise.all(promises).then(() => {
});*/
  console.log(results);
}

init();
```

- Ejemplo de uso típico y extendido de promesas

```
const fetch = require('node-fetch');

/*fetch(url)
  .then(response => response.json())
  .then(data => console.log(data));*/

async function get(url) {
  const response = await fetch(url);
  const data = await response.json();
  return data;
  /*return fetch(url)
    .then(response => response.json())
    .then(data => data);*/
};

async function init(){
  const url = 'https://reqres.in/api/users?page=2';
  const data = await get('https://reqres.in/api/users?page=2');
  const data2 = await get('https://reqres.in/api/users?page=3');
  console.log(data);
```

```
21    console.log(data2);
22  }
23
24  init();
```

3 Ejercicios

3.1 Creacion de funciones síncronas y asíncronas

3.2 Fetch

- fetch
- Fetch 2
- Con then-catch
- Con Async-await

3.3 FileSystem

- Usando lectura Sync
- Usando lectura Async

4 Entregables

4.1 En clase

- Ejecutar y modificar los códigos de ejemplo
- Crear algún flujo síncrono y asíncrono
- Leer un fichero de manera síncrona y asíncrona

4.2 Tarea