

EXERCISES QUE TRABAJAREMOS EN EL CUE

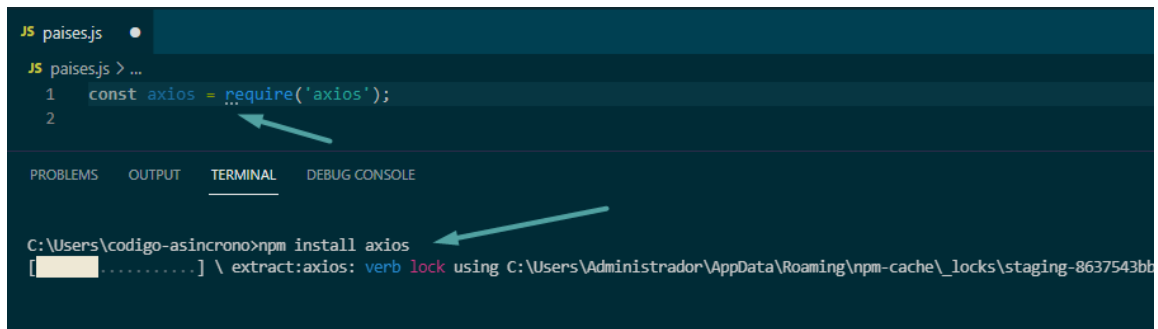
- EXERCISE 1: INSTALANDO AXIOS Y UTILIZANDO MÉTODO GET ()
- EXERCISE 2: UTILIZANDO MÓDULO FS JUNTO CON AXIOS.
- EXERCISE 3: PRACTICANDO EL USO DE CÓDIGO ASÍNCRONO CON UNA API.

EXERCISE 1: INSTALANDO AXIOS Y UTILIZANDO MÉTODO GET ()

Considerando otro caso de una situación real, donde las tareas pueden tomar más tiempo de lo normal, veamos qué es lo que pasa al momento de consultar una API.

Para ello haremos uso del módulo axios, el cual nos permite realizar peticiones http de forma más simplificada, y emplearemos la API Restcountries, que entrega información sobre todos los países del mundo.

Instala **axios** con el comando **"npm install axios"**, y crea un nuevo archivo países.js. Luego, importamos **axios**.



```
JS países.js
JS países.js > ...
1  const axios = require('axios');
2
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
C:\Users\codigo-asincrono>npm install axios
[redacted] \ extract:axios: verb lock using C:\Users\Administrador\AppData\Roaming\npm-cache\_locks\staging-8637543bb
```

Ahora usaremos el método **get()** de Axios, para así poder hacer la consulta a la URL <https://restcountries.eu/rest/v2/all>, la cual nos devolverá una lista de todos los países con sus respectivos datos. Axios entrega un objeto lleno de información útil, que incluye la respuesta que obtuvo de la URL en la propiedad data.

```
JS paises.js x
JS paises.js > ...
1  const axios = require('axios');
2
3  axios.get('https://restcountries.eu/rest/v2/all').then( respuesta => {
4
5      console.log(respuesta.data)
6
7  });
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
nativeName: 'Falkland Islands',
numericCode: '238',
currencies: [ [Object] ],
languages: [ [Object] ],
translations: {
  de: 'Falklandinseln',
  es: 'Islas Malvinas',
  fr: 'Îles Malouines',
  ja: 'フォークランド (マルビナス) 諸島',
  it: 'Isole Falkland o Isole Malvine',
  br: 'Ilhas Malvinas',
  pt: 'Ilhas Falkland',
  nl: 'Falklandeilanden [Islas Malvinas]',
  hr: 'Falklandski Otoci',
  fa: 'دنیلکل اف ری ارج'
},
flag: 'https://restcountries.eu/data/flk.svg',
regionalBlocs: [ [Object] ],
cioc: ''
},
{
  name: 'Faroe Islands',
  topLevelDomain: [ '.fo' ],
  alpha2Code: 'FO',
```

O también podemos extraer el contenido de “data”, directamente utilizando **Destructuring**.

```
JS paises.js x
JS paises.js > ...
1  const axios = require('axios');
2
3  axios.get('https://restcountries.eu/rest/v2/all').then( respuesta => {
4
5      const { data } = respuesta;
6      console.log(data)
7
8  });
```

También podemos consultar por un país específico, esto se hace utilizando la URL <https://restcountries.eu/rest/v2/name/{name}>, donde reemplazamos {name} por el nombre del país que queremos buscar, pero esta vez utilizando la sintaxis **async-await**.

```
JS países.js x
JS países.js > [0] obtenerPais
1  const axios = require('axios');
2
3  const obtenerPais = async () => {
4
5      const { data } = await axios.get('https://restcountries.eu/rest/v2/name/chile');
6      console.log(data);
7  }
8
9  obtenerPais();

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\codigo-asincrono>node países.js
[
  {
    name: 'Chile',
    topLevelDomain: [ '.cl' ],
    alpha2Code: 'CL',
    alpha3Code: 'CHL',
    callingCodes: [ '56' ],
    capital: 'Santiago',
    altSpellings: [ 'CL', 'Republic of Chile', 'República de Chile' ],
    region: 'Americas',
    subregion: 'South America',
    population: 18191900,
    latlng: [ -30, -71 ],
    demonym: 'Chilean',
    area: 756102,
    gini: 52.1,
    timezones: [ 'UTC-06:00', 'UTC-04:00' ],
    borders: [ 'ARG', 'BOL', 'PER' ],
    nativeName: 'Chile'
  }
]
```

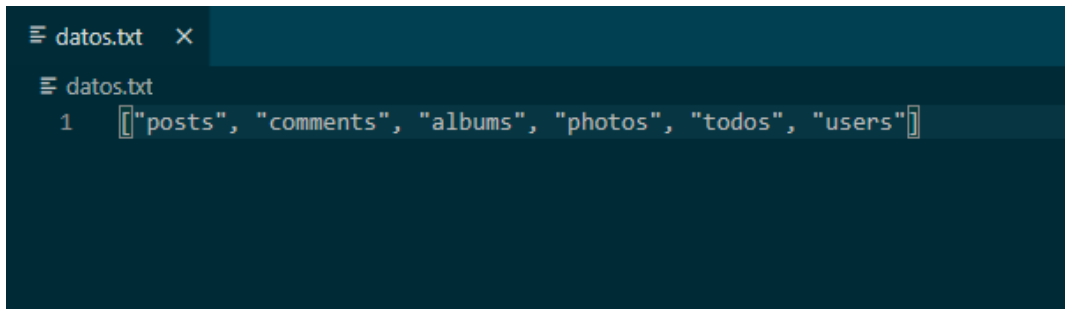
Ya estamos observando cómo es utilizar código asíncrono en situaciones reales, y, además, logrando crear código limpio, sabiendo exactamente cuál será el orden del resultado de la ejecución de tu programa.

EXERCISE 2: UTILIZANDO MÓDULO FS JUNTO CON AXIOS

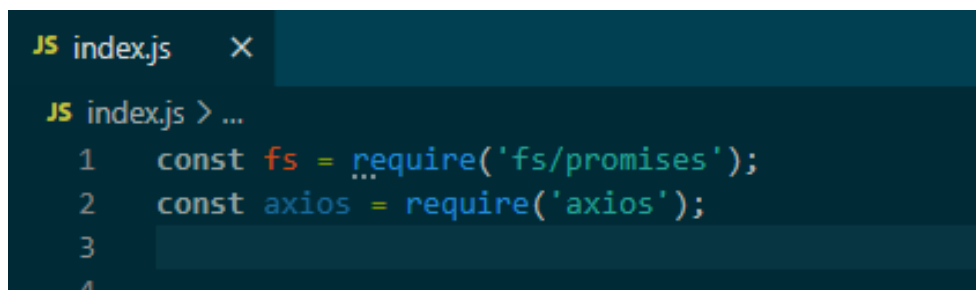
Para continuar con nuestra práctica en el uso de promesas y `async await`, seguiremos utilizando los módulos `fs` y `axios`.

También emplearemos una **API** llamada jsonplaceholder, que proporciona un juego de datos para prototipar aplicaciones. Ésta ofrece información falsa de posteos (posts), comentarios (comments), álbumes, fotos (photos), lista de tareas (todos), y usuarios (users).

Crearemos una nueva carpeta, un archivo `index.js`, y un archivo `datos.txt`. Dentro del archivo `datos.txt`, generaremos un Array en formato json, y sus ítems serán los tipos de información que jsonplaceholder entrega: `["posts", "comments", "albums", "photos", "todos", "users"]`

A screenshot of a code editor with a dark theme. The top bar shows a file named 'datos.txt' with a close button. The editor area shows the same file name and a single line of code: `["posts", "comments", "albums", "photos", "todos", "users"]`. The cursor is at the end of the line.

Instalaremos `axios` en nuestro proyecto con el comando `npm install axios`, y luego requeriremos `axios` y el módulo `fs` en nuestro archivo `index.js`.

A screenshot of a code editor with a dark theme. The top bar shows a file named 'index.js' with a close button. The editor area shows the same file name and the following code:

```
1 const fs = require('fs/promises');
2 const axios = require('axios');
3
4
```

Para que nuestro ejercicio sea más interesante, primero accederemos al sistema de archivos para revisar el contenido de `datos.txt`, y lo guardaremos en una variable. Ya que dicho contenido extraído del archivo `datos.txt` está en formato json, podemos leer esta data para obtener un Array, y guardarlo en una variable. Luego, crearemos una función que nos permita acceder de manera aleatoria a alguno de los ítems guardados en el Array, y con ese resultado, haremos uso de `axios` para obtener la data desde la [API](#) de jsonplaceholder.

Crearemos una función que genere un número al azar dependiendo del largo del Array de datos, y le restamos 1 para evitar acceder a alguna propiedad que no existe.

```
JS index.js X
JS index.js > ...
1  const fs = require('fs/promises');
2  const axios = require('axios');
3
4  const numeroAzar = (numeroMaximo) => Math.floor(Math.random() * numeroMaximo - 1);
5
6
7
8
```

Ahora, utilizamos el método `readFile()` para acceder a la información contenida en el archivo `datos.txt`. Guardaremos su valor en una variable, y esta vez no pasaremos una opción para codificar el archivo, pero tendremos que utilizar el método `JSON.parse` para convertir nuestra data a un Array.

```
JS index.js X
JS index.js > leerArchivo > arrayDatos
1  const fs = require('fs/promises');
2  const axios = require('axios');
3
4  const numeroAzar = (numeroMaximo) => Math.floor(Math.random() * numeroMaximo - 1);
5
6  const leerArchivo = async () => {
7    const datos = await fs.readFile('datos.txt');
8    const arrayDatos = JSON.parse(datos);
9    console.log(arrayDatos);
10 }
11
12 leerArchivo();
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
C:\Users\codigo-asincrono-ejercicios>node index.js
[ 'posts', 'comments', 'albums', 'photos', 'todos', 'users' ]
C:\Users\codigo-asincrono-ejercicios>
```

Ahora que ya tenemos nuestro Array de datos, utilizaremos el método `Math.abs()` para acceder a un ítem de éste, y como argumento, pasaremos la cantidad total de ítems. Además, agregaremos la función `Math.abs()` a nuestro número, para prevenir el caso de que exista un negativo.

```

1  const fs = require('fs/promises');
2  const axios = require('axios');
3
4  const numeroAzar = (numeroMaximo) => Math.floor(Math.random() * numeroMaximo - 1);
5
6  const leerArchivo = async () => {
7    const datos = await fs.readFile('datos.txt');
8    const arrayDatos = JSON.parse(datos);
9
10   const numeroArray = Math.abs(numeroAzar(arrayDatos.length));
11
12   console.log(arrayDatos[numeroArray]);
13 }
14
15 leerArchivo();

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```

C:\Users\codigo-asincrono-ejercicios>node index.js
comments
C:\Users\codigo-asincrono-ejercicios>node index.js
comments
C:\Users\codigo-asincrono-ejercicios>node index.js
posts
C:\Users\codigo-asincrono-ejercicios>node index.js
albums
C:\Users\codigo-asincrono-ejercicios>node index.js
photos
C:\Users\codigo-asincrono-ejercicios>

```

Ahora utilizamos Axios, junto con la siguiente URL <https://jsonplaceholder.typicode.com/{datos}>, donde {datos} será reemplazado por nuestro valor al azar.

```

1  const fs = require('fs/promises');
2  const axios = require('axios');
3
4  const numeroAzar = (numeroMaximo) => Math.floor(Math.random() * numeroMaximo - 1);
5
6  const leerArchivo = async () => {
7    const datos = await fs.readFile('datos.txt');
8    const arrayDatos = JSON.parse(datos);
9
10   const numeroArray = Math.abs(numeroAzar(arrayDatos.length));
11
12   const valorBusqueda = arrayDatos[numeroArray];
13
14   const { data } = await axios.get(`https://jsonplaceholder.typicode.com/${valorBusqueda}`);
15
16   console.log(data);
17 }
18
19 leerArchivo();

```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```

{
  "omnis facilis nam ipsum natus sint similique omnis"
},
{
  "userId: 10,
  id: 89,
  title: 'id minus libero illum nam ad officis',
  body: 'eaum voluptates facere provident blanditiis velit laboriosam\n' +
    'pariatur accusamus odio saepe\n' +
    'cumque dolor qui a dicta ab doloribus consequatur omnis\n' +
    'corporis cupiditate eaque assumenda ad nesciunt'
},
{
  "userId: 10,
  id: 96,
  title: 'consect etiam enim aut cupiditate est namque ut conu'
}

```

Al volver a ejecutar nuestro programa, obtenemos, dependiendo del azar, la siguiente información:

```

    },
    {
      postId: 17,
      id: 85,
      name: 'non voluptas cum est quis aut consectetur nam',
      email: 'Alexzander_Davis@eduardo.name',
      body: 'quisquam incidunt dolores sint qui doloribus provident\n' +
        'vel cupiditate deleniti alias voluptatem placeat ad\n' +
        'ut dolorem illum unde iure quis libero neque\n' +
        'ea et distinctio id'
    },
    {
      postId: 18,
      id: 86,
      name: 'suscipit est sunt vel illum sint',
      email: 'Jacquelyn@krista.info',
      body: 'eum culpa debitis sint\n' +
        'eaque quia magni laudantium qui neque voluptas\n' +
        'voluptatem qui molestiae vel earum est ratione excepturi\n' +
        'sit aut explicabo et repudiandae ut perspiciatis'
    },
    {
      postId: 18,
      id: 87,
      name: 'dolor asperiores autem et omnis quasi nobis',
      email: 'Grover_Volkman@coty.tv',
      body: 'assumenda corporis architecto repudiandae omnis qui et odit\n' +
        'perferendis velit enim\n' +
        'et quia reiciendis sint\n'
    }
  ],
  {
    postId: 18,
    id: 87,
    name: 'dolor asperiores autem et omnis quasi nobis',
    email: 'Grover_Volkman@coty.tv',
    body: 'assumenda corporis architecto repudiandae omnis qui et odit\n' +
      'perferendis velit enim\n' +
      'et quia reiciendis sint\n'
  }
]

```

Hasta ahora hemos estado utilizando la sintaxis **async-await** para crear nuestro programa, veamos cómo quedaría si decidimos utilizar promesas.

```

JS index.js  X
JS index.js > ...
1  const fs = require('fs/promises');
2  const axios = require('axios');
3
4  const numeroAzar = (numeroMaximo) => Math.floor(Math.random() * numeroMaximo - 1);
5
6  fs.readFile('datos.txt')
7    .then( datos => {
8      const arrayDatos = JSON.parse(datos);
9
10     const numeroArray = Math.abs(numeroAzar(arrayDatos.length));
11
12     const valorBusqueda = arrayDatos[numeroArray];
13
14     return axios.get( https://jsonplaceholder.typicode.com/\${valorBusqueda} );
15   })
16   .then( respuesta => {
17     const { data } = respuesta;
18
19     console.log(data);
20   });
21

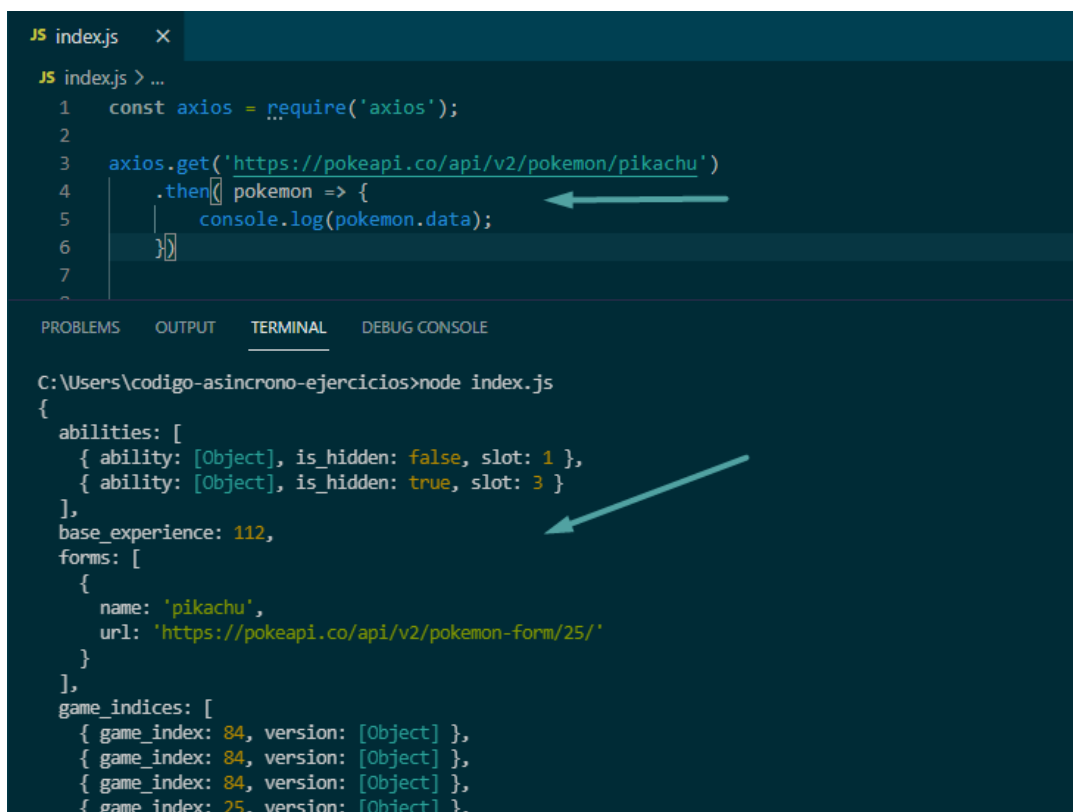
```

Si lo puedes notar, el resultado del programa es el mismo, pero la sintaxis se ve un poco menos limpia.

EXERCISE 3: PRACTICANDO EL USO DE CÓDIGO ASÍNCRONO CON UNA API

Como último ejercicio para practicar el uso de código asíncrono, exploraremos la API de Pokémon. Ésta contiene la información de los Pokémon de todas las temporadas, junto a la información de cada uno. Puedes buscar la documentación en la URL <https://pokeapi.co/>, y así conocer todas las rutas que puedes consultar para obtener datos importantes.

Para este ejercicio utilizaremos la siguiente URL: <https://pokeapi.co/api/v2/pokemon/{nombre}>, donde reemplazaremos `{nombre}` por el nombre del Pokémon que queramos buscar. Emplearemos Axios para hacer la llamada, y esta vez, partiremos con el uso de promesas y `then()`.



```
JS index.js x
JS index.js > ...
1  const axios = require('axios');
2
3  axios.get('https://pokeapi.co/api/v2/pokemon/pikachu')
4    .then(pokemon => {
5      console.log(pokemon.data);
6    })
7
8
9
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
C:\Users\codigo-asincrono-ejercicios>node index.js
{
  abilities: [
    { ability: [Object], is_hidden: false, slot: 1 },
    { ability: [Object], is_hidden: true, slot: 3 }
  ],
  base_experience: 112,
  forms: [
    {
      name: 'pikachu',
      url: 'https://pokeapi.co/api/v2/pokemon-form/25/'
    }
  ],
  game_indices: [
    { game_index: 84, version: [Object] },
    { game_index: 84, version: [Object] },
    { game_index: 84, version: [Object] },
    { game_index: 25, version: [Object] },
  ]
}
```

Ahora agregamos el método `catch()`, para controlar un error en caso de que éste ocurra, por ejemplo, cuando nos equivocamos al copiar la URL.


```
JS index.js x
JS index.js > ...
1  const axios = require('axios');
2
3  axios.get('https://poke.co/api/v2/pokemon/pikachu')
4    .then( pokemon => {
5      console.log(pokemon.data);
6    })
7    .catch( error => {
8      console.log(error);
9    })
10
11
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
C:\Users\codigo-asincrono-ejercicios>node index.js
Error: connect ECONNREFUSED 52.128.23.153:443
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1146:16) {
  errno: -4078,
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '52.128.23.153',
  port: 443,
  config: {
    url: 'https://poke.co/api/v2/pokemon/pikachu',
    method: 'get',
    headers: {
      Accept: 'application/json, text/plain, */*',
      'User-Agent': 'axios/0.21.1'
    },
    transformRequest: [ [Function: transformRequest] ],
    transformResponse: [ [Function: transformResponse] ],
    timeout: 0
  }
}
```

Si quisiéramos utilizar la sintaxis **async-await**, podríamos escribir nuestro código de la siguiente forma:

```
JS index.js x
JS index.js > ...
1  const axios = require('axios');
2
3  const obtenerPokemon = async () => {
4    const { data } = await axios.get('https://pokeapi.co/api/v2/pokemon/pikachu');
5    console.log(data);
6  }
7
8  obtenerPokemon();
9
10
```

Y, por último, añadiendo control de errores con un bloque **try catch**.

```
JS index.js X
JS index.js > [E] obtenerPokemon
1  const axios = require('axios');
2
3  const obtenerPokemon = async () => {
4    try {
5      const { data } = await axios.get('https://pokeapi.co/api/v2/pokemon/pikachu');
6      console.log(data);
7    } catch(error){
8      console.log(error);
9    }
10 }
11
```