JavaScript Await Async son dos de las palabras reservas que se usan en JavaScript ES7 para simplificar la gestión de la programación concurrente y que nos puede ayudar a simplificar el manejo de peticiones AJAX y la gestión de promesas. Vamos a ver un ejemplo sencillo ,para ello partiremos de un servidor de Node.js que nos devuelve una lista de facturas.

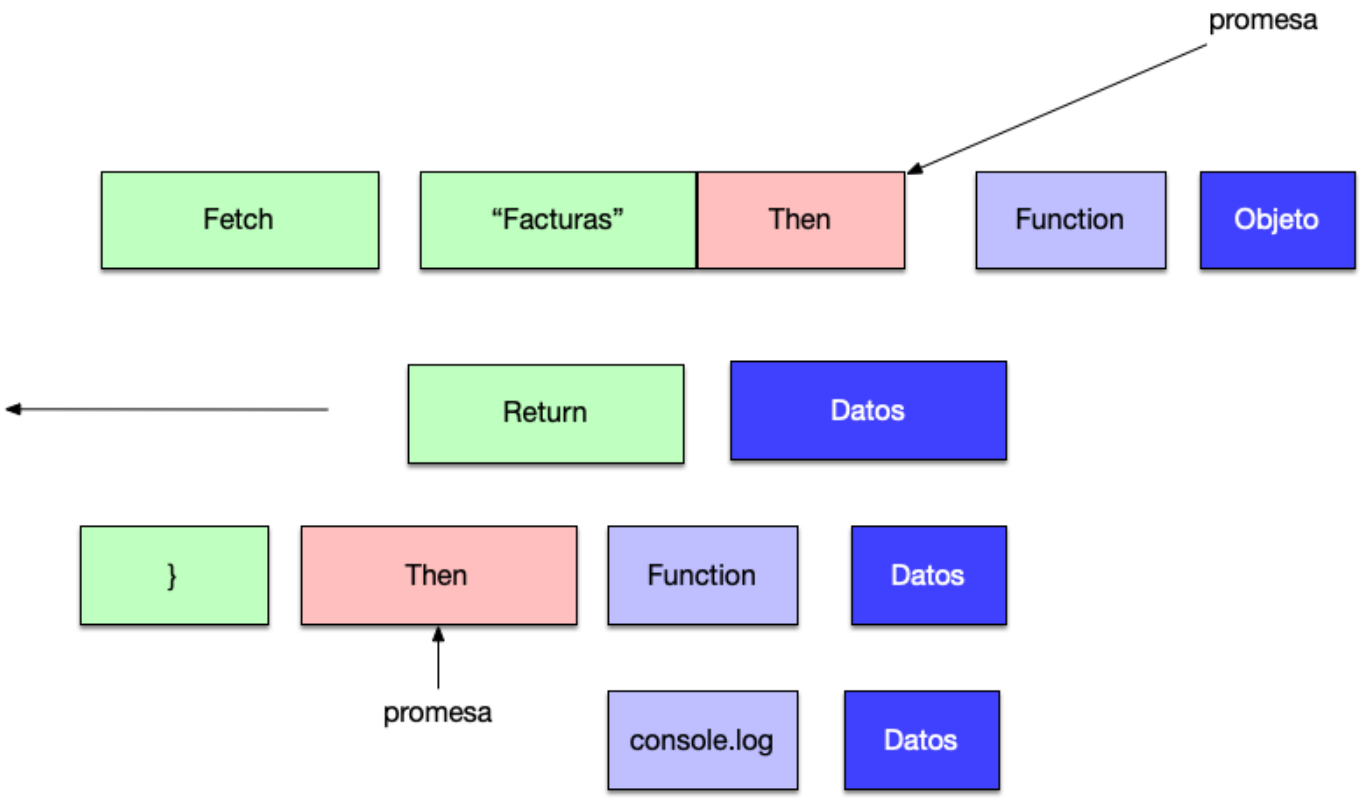
```
var express = require('express');
var app = express();
app.use(express.static('publica'))


let lista = [];
lista.push({
  'id': '1',
  'concepto': 'ordenador',
  'importe': '500'
});
lista.push({
  'id': '2',
  'concepto': 'tablet',
  'importe': '300'
});

app.get('/facturas', function(req, res) {
  res.send(lista);
});

app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

Lo único que tenemos es que disponer de una página html que haga una petición AJAX al servidor y se traiga los datos de las Facturas.

```
<html>

<head>

  <script type="text/javascript">
     function cargaAjax() {

        fetch("facturas").then(function(promesa) {

          return promesa.json();

        }).then(function(datos) {

            console.log(datos);
        })
    }


  </script>

</head>

<body>
  <p>mipagina</p>
  <table id="mitabla">

  </table>


  <input type="button" onclick="cargaAjax()" value="aceptar" />
</body>
```

```
</html>
```

En este caso estamos ante una petición muy muy sencilla que se realiza usando el API de Fetch de JavaScript ES6 que esta orientado al manejo de promesas. Solicitamos al servidor las facturas y el API de Fetch enlazara una promesa. Esta promesa contiene los datos en JSON que de igual manera se pueden procesar como una promesa. En principio parece un poco engorroso comparado con una petición clásica de jQuery.




Sin embargo esta basada en la idea de promesas de jQuery

# JavaScript Async Await

Es aquí donde las nuevas palabras reservadas de Async y Await nos pueden ayudar a simplificar el código de forma clara. La palabra Async sirve definir que una función es asincrona. Por lo tanto lo tendremos que poner a nivel de la función:

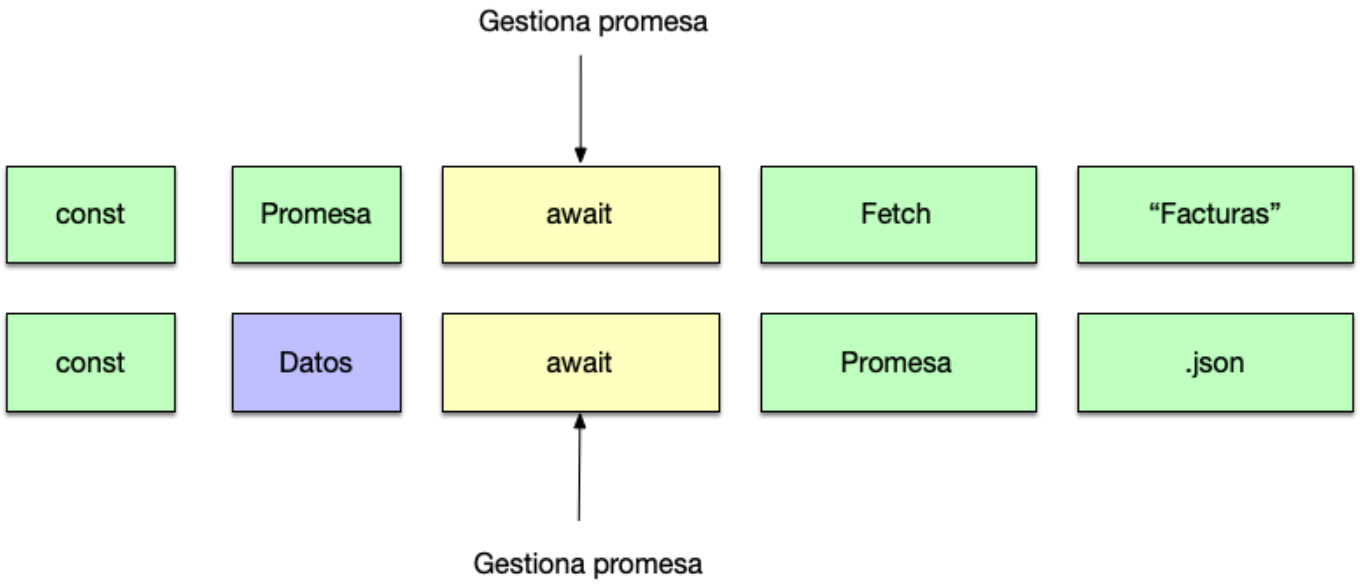
```
<html>

<head>
  <script type="text/javascript">
    async  function cargaAjax() {



    }
  </script>
</head>

<body>
  <input type="button" onclick="cargaAjax()" value="aceptar" />
</body>

</html>
```

Por lo tanto el primer paso es modificar el código para que soporte JavaScript Async Await. En este caso lo que hacemos es marcar la función con la palabra reservada async. Esto ya hace que cualquier navegador moderno de JavaScript considere la función asíncrona. Nos queda aplicar await a cada una de los métodos que anteriormente soportaban el método then.

```
<html>

<head>
  <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <script src="lodash.js"></script>
  <script type="text/javascript">
    async  function cargaAjax() {
```

```
        const promesa=await fetch("facturas");
        const datos= await promesa.json();
        console.log(datos);
    }
  </script>

</head>

<body>
  <input type="button" onclick="cargaAjax()" value="aceptar" />
</body>

</html>
```

El código queda muy simplificado a nivel de gestión de promesas. El método await se encarga de añadir la sintaxis sugar necesaria para procesar lo que anteriormente hacían los métodos then.




Poco a poco tenemos que irnos acostumbrando a usar más las capacidades modernas de

JavaScript si nuestro navegador lo soporta o usamos un transcompilador tipo babel.

## Otros artículos relacionados

- jQuery Deferred
- Promesas y chaining
- Promise vs Observable

Cecilio Álvarez Caules
Cecilio Álvarez Caules Oracle Java Certified Architech