



- Back-End(<https://imasters.com.br/back-end>)
- Mobile(<https://imasters.com.br/mobile>)
- Front End(<https://imasters.com.br/front-end>)
- DevSecOps(<https://imasters.com.br/devsecops>)
- Design & UX(<https://imasters.com.br/design-ux>)
- Data(<https://imasters.com.br/data>)
- APIs e Microsserviços(<https://imasters.com.br/apis-microsservicos>)
- IoT e Makers(<https://imasters.com.br/iot-makers>)

DESENVOLVIMENTO

4 MAI, 2018

Funções assíncronas e retornos: como o Async/Await tornaram o código mais legível

100 visualizações



(<https://www.facebook.com/sharer?u=https://imasters.com.br/desenvolvimento/funcoes-assincronas-e-retornos-como-o-async-await-tornaram-o-codigo-mais-legivel>)



(<https://twitter.com/share?url=https://imasters.com.br/desenvolvimento/funcoes-assincronas-e-retornos-como-o-async-await-tornaram-o-codigo-mais-legivel>)



(<https://www.linkedin.com/shareArticle?url=https://imasters.com.br/desenvolvimento/funcoes-assincronas-e-retornos-como-o-async-await-tornaram-o-codigo-mais-legivel>)

COMPARTILHE!

LUCAS SANTOS
(<https://imasters.com.br/perfil/lucassantos>)
Tem 26 artigos publicados com 29053 visualizações desde 2017



PUBLICIDADE

LUCAS SANTOS (<https://imasters.com.br/perfil/lucassantos>)

26

Estudante de Ciências da Computação pela UFABC. Desenvolvedor desde 2011, trabalha com infraestrutura e arquitetura de aplicações de alta disponibilidade. Participante ativo de comunidades, organizador do ABCDev, membro do Core Team do Training Center e moderador no fórum iMasters.

LEIA MAIS (<https://imasters.com.br/perfil/lucassantos>)

2 JUL, 2020

Design Patterns com JavaScript & TypeScript: Padrões Estruturais (<https://imasters.com.br/javascript/design-patterns-com-javascript-typescript-padroes-estruturais>)

8 ABR, 2020

O Lado Escuro do JavaScript: O retorno (<https://imasters.com.br/javascript/o-lado-escuro-do-javascript-o-retorno>)

9 MAR, 2020

Design Patterns com JavaScript & TypeScript (<https://imasters.com.br/devsecops/design-patterns-com-javascript-typescript>)

Durante os anos de atualização do ECMAScript, o JavaScript veio recebendo uma série de funcionalidades novas que facilitaram muito a vida do desenvolvedor. O suporte a [Promises do ES6](https://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects) (<https://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects>) criou um mundo completamente novo, onde poderíamos tirar proveito do assincronismo da linguagem e de todo o seu ferramental performático.

O problema com as Promises era o chamado “promise hell”, assim como o tão conhecido callback hell das versões anteriores do JavaScript, esse novo problema surge quando começamos a encadear muitas chamadas then nas nossas resoluções de Promises, por exemplo:

```
1 | usuario.obterPerfil()  
2 |   .then((perfil) => {  
3 |     usuario.obterMidias(perfil)  
4 |     .then((midias) => {  
5 |       midias.obterDados()  
6 |       .then((dados) => {...})  
7 |     })  
8 |   })
```

Percebe como a legibilidade fica complicada? Para resolver esse problema, o ES2017 implementou o que hoje chamamos de Async/Await, que deixou o código muito mais legível. Por exemplo, o código anterior ficaria desta forma:

```
1 | const perfil = await usuario.obterPerfil()  
2 | const midiasPerfil = await usuario.obterMidias(perfil)  
3 | const midiasDados = await midias.obterDados()
```

Porém, agora ganhamos um novo tipo de função, as **funções assíncronas**. Em suma, uma função assíncrona contém a palavra-chave async em sua assinatura e, por padrão, retorna uma Promise. Esses retornos ficam um pouco confusos quando temos que retornar exatamente o resultado de uma Promise para a função anterior, então vamos entender um pouco como podemos utilizar o async/await junto com o return, e veja que escolher a forma certa é muito importante.

Vamos começar com uma função assíncrona simples:

```
1 | async function espereEDecida() {  
2 |   // Vamos esperar um segundo primeiro  
3 |   await new Promise((resolve) => setTimeout(resolve, 1000))  
4 |  
5 |   // Vamos fazer uma jogada aleatória, com 50% de chance para true ou false  
6 |   const sim = Boolean(Math.round(Math.random()))  
7 |  
8 |   if (sim) return 'yeah!'  
9 |   throw Error('Ops!')  
10 | }
```

Basicamente, essa função dá uma resposta que pode ser uma resolução ou um erro dentro de um segundo depois de sua chamada.

A partir daí, temos quatro formas de poder chamar essa função, e as quatro se comportam de formas completamente diferentes:

1. Uma chamada simples

Podemos realizar somente uma chamada desse tipo:

```
1 | async function f() {  
2 |   try {  
3 |     espereEDecida()  
4 |   } catch (err) { return 'peguei' }  
5 | }
```

Neste exemplo, se chamarmos a função f, a Promise que for retornada **sempre vai ser resolvida para undefined**, sem nenhum tipo de espera. Isso acontece porque não estamos sequer tratando o resultado retornado por espereEDecida, estamos apenas invocando a função e colocando sua chamada na fila de execuções. Uma vez que ela é executada, nós não tratamos o código de nenhuma maneira, então não fazemos nada com ela.

Esse tipo de código é muito raro de ser utilizado, e geralmente é considerado um erro, uma vez que não estamos fazendo nada com o código, mas, se o retorno desta função fosse uma Promise pura ao invés de uma string, poderíamos utilizar esse tipo de chamada para executar uma Promise interna sem nos importar com o resultado.

2. Somente Await

Agora podemos fazer desta forma:

```
1 | async function f() {  
2 |   try {  
3 |     await espereEDecida()  
4 |   } catch (err) { return 'peguei' }  
5 | }
```

Se chamarmos nossa função agora, a Promise interna vai **sempre esperar um segundo** e depois vai seguir para a nossa jogada de sorte que **pode ser resolvida como undefined ou então vai ser resolvida com “peguei”**.

Isso vai acontecer porque estamos esperando o resultado de `espereEDecida()`. Se ela for rejeitada, temos um throw que será pego pelo nosso bloco catch na função `f`. No entanto, se ela for resolvida com sucesso, teremos somente `undefined` como retorno porque não estamos armazenando o valor da Promise em lugar nenhum.

3. Retornando uma Promise

A terceira forma seria algo assim:

```
1 | async function f() {  
2 |   try {  
3 |     return espereEDecida()  
4 |   } catch (err) { return 'peguei' }  
5 | }
```

Agora, se chamarmos a função `f` neste caso, teremos um resultado interessante, pois nossa Promise de `espereEDecida()` vai **sempre esperar um segundo e ser resolvida ou rejeitada com ‘ops!’**, como seria o esperado dela, mas o interessante é que nosso bloco catch não vai nunca ser executado, pois estamos delegando o retorno dessa Promise para quem está chamando a função `f`, e não para ela própria.

4. Retornando com Await

Este é o caso em que juntamos todos os anteriores em um, ficando alguma coisa assim:

```
1 | async function f() {  
2 |   try {  
3 |     return await espereEDecida()  
4 |   } catch (err) { return 'peguei' }  
5 | }
```

Aqui teremos algo muito parecido com o caso anterior. Se chamarmos a função `f`, a Promise de dentro vai **esperar um segundo e ser resolvida ou rejeitada com ‘peguei’**, veja que agora a rejeição da Promise não está em quem está invocando a mesma, mas sim dentro da própria função.

Isso acontece porque estamos esperando o resultado de `espereEDecida()`. Se ela for rejeitada, então essa rejeição se transformará em um throw que será pego pelo nosso catch, dando o retorno de ‘peguei’. Agora, se ela for resolvida, o retorno da Promise será ‘yeah!’, que será retornado pela função `f` para seu invocador.



Conectividade em Cloud

Supere seus desafios de escalabilidade com o Equinix Cloud Exchange Fabric™.

Ficou um pouco confuso, não é? Vamos simplificar. Este passo pode ser pensado como sendo uma sequência de duas execuções:

```
1 async function f() {  
2   try {  
3     // Aqui esperamos o resultado da função, ela pode retornar 'Yeah!' ou 'Ops!'  
4     const resultado = await espereEDecida()  
5  
6     // Se a função espereEDecida() rejeitar, teremos um throw  
7     // Ele será pego pelo nosso catch abaixo  
8     // Caso contrário, vamos retornar o resultado propriamente dito, que será 'Yeah!'  
9     return resultado  
10  } catch (err) { return 'peguei' }  
11 }
```

O caso 4 tem uma peculiaridade, pois sem os blocos try/catch, um return seguido de await é completamente redundante (já até criaram uma [regra do ESLint](https://github.com/eslint/eslint/blob/master/docs/rules/no-return-await.md) (<https://github.com/eslint/eslint/blob/master/docs/rules/no-return-await.md>) para estes casos). Isso porque o retorno de uma função async é automaticamente envolto em um Promise.resolve, como nesta [página da MDN](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/funcoes_assincronas) (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/funcoes_assincronas). Ou seja, se fizermos algo assim:

```
1 async function pegarDadosProcessados(url) {  
2   let v;  
3   try {  
4     v = await baixarDados(url);  
5   } catch(e) {  
6     v = await baixarDadosReservas(url);  
7   }  
8   return processarDadosNoWorker(v);  
9 }
```

E retornarmos a Promise diretamente no final da função, teremos implicitamente um Promise.resolve(processarDadosNoWorker(v)).

Funções assíncronas são bastante comuns atualmente, mas algumas dicas como estas não são muito explícitas e exigem um pouco de testes e erros para se descobrir. Portanto, acredito que saímos daqui com um pouco mais de conhecimento sobre o que as funções assíncronas fazem e como elas funcionam no JavaScript.



De 0 a 10, o quanto você recomendaria este artigo para um amigo?



ARTIGOS PUBLICADOS POR ESTE AUTOR



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
2 JUL, 2020



Design Patterns com JavaScript & TypeScript: Padrões Estruturais (<https://imasters.com.br/javascript/design-patterns-com-javascript-typescript-padroes-estruturais>)



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
8 ABR, 2020



O Lado Escuro do JavaScript: O retorno (<https://imasters.com.br/javascript/o-lado-escuro-do-javascript-o-retorno>)



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
9 MAR, 2020



Design Patterns com JavaScript & TypeScript (<https://imasters.com.br/devsecops/design-patterns-com-javascript-typescript>)



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
12 FEV, 2020



Entendendo o ciclo de publicação do Node.js (<https://imasters.com.br/javascript/entendendo-o-ciclo-de-publicacao-do-node-js>)



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
14 JAN, 2020



JavaScript em 2020: O que esperar (<https://imasters.com.br/javascript/javascript-em-2020-o-que-esperar>)



LUCAS SANTOS ([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))
13 DEZ, 2019



Optional Chaining e null coalescing operator com TypeScript (<https://imasters.com.br/javascript/optional-chaining-e-null-coalescing-operator-com-typescript>)

Qual é seu nível de satisfação com a
experiência geral nesta página da Web?



Muito insatisfeito

Muito satisfeito



SAIBA MAIS
([HTTPS://IMASTERS.COM.BR/PERFIL/LUCASSANTOS](https://imasters.com.br/perfil/lucassantos))

Lucas Santos



(https://twitter.com/_StaticVoid)



(<mailto:lhs.santoss@gmail.com>)

26 Artigo(s)

Estudante de Ciências da Computação pela UFABC. Desenvolvedor desde 2011, trabalha com infraestrutura e arquitetura de aplicações de alta disponibilidade. Participante ativo de comunidades, organizador do ABCDev, membro do Core Team do Training Center e moderador no fórum iMasters.



(<http://clicklogger.rm.uol.com.br/?>

<http://www.zarpsystem.com.br/>
prd=16&grp=src:34;chn:118;cpg:imasters2019;&msr=Cliques%20de%20Origem:1&oper=11&redir=https://uolhost.uol



Este projeto é apoiado pelas empresas



(<https://imasters.tech/>)



(<http://www.w3c.br>)

ASSINE NOSSA Newsletter

Fique em dia com as novidades do iMasters! Assine nossa newsletter e receba conteúdos especiais curados por nossa equipe



Qual é o seu e-mail?

ASSINAR



[SOBRE O IMASTERS \(HTTPS://IMASTERS.COM.BR/P/SOBRE-O-IMASTERS\)](https://imasters.com.br/p/sobre-o-imasters)

[POLÍTICA DE PRIVACIDADE \(HTTPS://IMASTERS.COM.BR/P/POLITICA-DE-PRIVACIDADE\)](https://imasters.com.br/p/politica-de-privacidade)

[FALE CONOSCO \(HTTPS://IMASTERS.COM.BR/FALE-CONOSCO/\)](https://imasters.com.br/faq)

[QUERO SER AUTOR \(HTTPS://IMASTERS.COM.BR/P/QUERO-SER-AUTOR\)](https://imasters.com.br/p/quero-ser-autor)

[FÓRUM \(HTTPS://FORUM.IMASTERS.COM.BR/\)](https://forum.imasters.com.br/)

[7MASTERS \(HTTPS://SETEMASTERS.IMASTERS.COM.BR/\)](https://setemasters.imasters.com.br/)

[AGENDA \(HTTPS://IMASTERS.COM.BR/AGENDA/\)](https://imasters.com.br/agenda/)

[IMASTERS.COM \(HTTPS://IMASTERS.COM/\)](https://imasters.com/)

0 comentários

Classificar por **Mais antigos**

Adicione um comentário...

Plugin de comentários do Facebook

Este projeto é oferecido pelas empresas



(http://bit.ly/2sB2idH)

(http://impulso.network/?
utm_source=imasters&utm_medium=site&utm_campaign=footer)

Este projeto é mantido e patrocinado pelas empresas



(http://www.alura.com.br)



(https://www.cielo.com.br/e-commerce/)

(http://www.dialhost.com.br?
utm_campaign=PatrocinioiMasters&utm_sour

(https://fiap.me/2C8SbRO)



(https://www.hostgator.com.br/?

utm_source=imasters&utm_medium=logo&utm_campaign=hostgator&utm_term=hospec



(https://www.idexo.com.br/)



(https://www.impacta.com.br/)



(http://www.ingrammicro.com.br/isv/)

(https://king.host/?
utm_source=parceiros&utm_medium=&utm_term=2019&utm_campaign=)

(http://lambda3.com.br/)



(https://www.locaweb.com.br/?

utm_campaign=eventos&utm_source=imasters&utm_medium=site&utm_content=locaweb



(http://bit.ly/2BXbpvx)



(https://www.userede.com.br/)

(https://www.schoolofnet.com/cursos/gratuitos/?
utm_source=imasters&utm_medium=patrocinio&utm_campaign=institucional_patrocinio_institucional&utm_content=institucional_patrocinio_imasters_link-institucional)

(https://developers.totvs.com/)