**Para saber mais: função do Node.js**

**Para saber mais: JavaScript assíncrono**

Para entender melhor sobre requisições GET no consumo de API recomendo fortemente que você assista o curso [JavaScript: Consumindo e tratando dados de uma API](https://cursos.alura.com.br/course/javascript-consumindo-tratando-dados-api), que aborda de forma mais detalhada como o JavaScript interpreta trechos de códigos e como podemos torna-los assíncrono. Além de explicar sobre fetch API e sobre termos que aparecem ao trabalhar com requisições.

Caso você já tenha assistido e gostaria de rever conceitos, você pode acompanhar o Elias Ribeiro em seu artigo [Começando com fetch no JavaScript](https://www.alura.com.br/artigos/comecando-com-fetch-no-javascript) e a Juliana Negreiros no Alura Mais sobre [JavaScript assíncrono e fetch](https://cursos.alura.com.br/extra/alura-mais/javascript-assincrono-e-fetch-c93).

A abordagem usada para a construção de funções assíncronas foi o “async await” e você pode ler mais sobre isso no artigo [Async/await no JavaScript: o que é e quando usar a programação assíncrona?](https://www.alura.com.br/artigos/async-await-no-javascript-o-que-e-e-quando-usar" \t "_blank) da Juliana Amoasei.

**Para saber mais: mergulhe no JavaScript**

Para individualizar elementos e conseguirmos selecioná-los no JavaScript para manipular o DOM, foram inseridos **data-attributes**. A Juliana Negreiros te explica melhor sobre esse termo em seu Alura Mais [Data Attributes do HTML5](https://cursos.alura.com.br/extra/alura-mais/data-attributes-do-html5-c109).

Após selecionar esses elementos com o **querySelector** gostaríamos de inserir um código HTML dentro deles. Com isso, construímos a função constroiCard() e usamos o **innerHTML** e atribuímos a ele um trecho de código em forma de string com o auxílio das **template strings** para torná-lo dinâmico. Você pode assistir mais sobre esse assunto no Alura Mais [Template String](https://cursos.alura.com.br/extra/alura-mais/template-string-c123) do Felipe Nascimento e também ir além: aprender mais formas de manipular strings, lendo o artigo [Strings com JavaScript: o que são e como manipulá-las](https://www.alura.com.br/artigos/strings-com-javascript-o-que-sao-e-como-manipular" \t "_blank) do André Bessa.

Com o intuito de tornar o código mais legível e separar funções e arquivos de acordo com a sua responsabilidade no projeto, utilizamos o **import** e **export** para aproveitar funções de outros arquivos como a listaVideos() que está localizada no arquivo conectaApi.js. Quer entender melhor como isso foi feito? Vêm com o Mario Souto em seu artigo [Como funciona o import e export do JavaScript?](https://www.alura.com.br/artigos/como-funciona-o-import-e-export-do-javascript).

Para conseguir aplicar a função constroiCard() em cada um dos itens que retornaram da lista de vídeos do servidor, aplicamos o método forEach (método abordado no curso [JavaScript: métodos de array](https://cursos.alura.com.br/course/javascript-metodos-array)). Dentro dele, foi construído uma função com uma sintaxe diferente, utilizando =>, que se assemelha a uma flecha. E não é só semelhança: o nome desses tipo de função é arrow function do inglês ‘função flecha’. Descubra mais sobre elas com o Felipe Nascimento no artigo [Conhecendo Arrow Functions](https://www.alura.com.br/artigos/conhecendo-arrow-functions).