Axios - Promise Based HTTP Client

[**1. Axios Introdução 1**](#_ihv636)

[**2. Instalação 1**](#_32hioqz)

[**3. Exemplos Básicos 1**](#_1hmsyys)

[**4. Solicitações POST 2**](#_41mghml)

[4.0 Formulários 3](#_2grqrue)

[**5. API Axios 4**](#_vx1227)

[**6. Instâncias Axios 4**](#_3fwokq0)

[**7. Configuração de requisição 5**](#_1v1yuxt)

[**8. Esquema de Resposta 9**](#_4f1mdlm)

[**9. Configurações de Padrões 9**](#_2u6wntf)

[**10. Interceptadores 10**](#_19c6y18)

[**11. Manipuladores de Erros 11**](#_3tbugp1)

[**12. Cancelamento 12**](#_28h4qwu)

[12.0 AbortController 12](#_nmf14n)

[12.1 CancelToken (DESCONTINUADO) 12](#_37m2jsg)

[**13. Corpo de Codificação de URL 13**](#_1mrcu09)

# **1. Axios Introdução**

Baseado em promises node.js para o navegador.

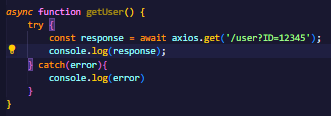
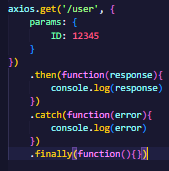
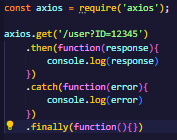
**Nota: no lado do servidor utiliza o http (módulo nativo node.js) e no cliente (navegador) utiliza XMLHttpRequest**.

# **2. Instalação**

(**DENTRO DA PASTA DO PROJETO**) No terminal, digitar: npm i axios.

# **3. Exemplos Básicos**

**Exemplo**:

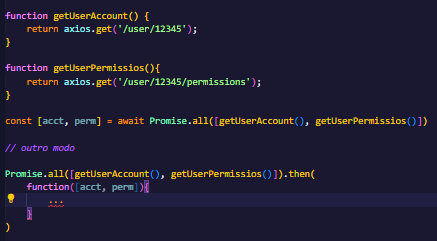


# **4. Solicitações POST**

**Exemplo: executando uma solicitação POST**.  

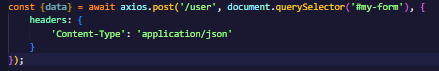

**entrada com os dados em formato JSON de nome e sobrenome do cliente**.

**Exemplo: execução de múltiplas solicitações simultâneas**.



**cria duas funções com solicitações get, que serão requeridas dentro da promise.all**.

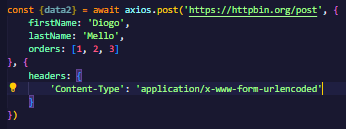
**Exemplo: postagem de formulários html com JSON**.



## **4.0 Formulários**

**Exemplo: Multiparte**.  


**Exemplo: Formulário codificado em URL**.



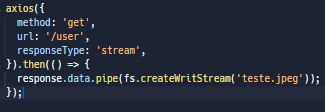
# **5. API Axios**

Requisições podem ser realizadas por meio de configurações no axios.

**Exemplo**:



**requisição post**.



**requisição get para imagem remota em node**.

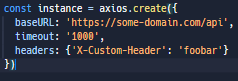


**Métodos de solicitação suportados**:

1. axios.request(config).
2. axios.get(url[config]).
3. axios.delete(url[config]).
4. axios.head(url[config]).
5. axios.options(url[config]).
6. axios.post(url[, data, config]).
7. axios.put(url[, data, config]).
8. axios.patch(url[, data, config]).

# **6. Instâncias Axios**

Exemplo: criação de instância com configuração customizada.



**Métodos de solicitação suportados**:

1. axios#request(config).
2. axios#get(url[config]).
3. axios#delete(url[config]).
4. axios#head(url[config]).
5. axios#options(url[config]).
6. axios#post(url[, data, config]).
7. axios#put(url[, data, config]).
8. axios#patch(url[, data, config]).
9. axios#getUri([config]).

# **7. Configuração de requisição**

**OBS: configuração de requisição de url é obrigatória**.

**Nota: se nenhum method for especificado, todas as requisições serão tratadas como get**.

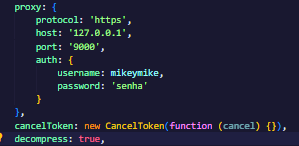
**Especificações**:

* **url** - servidor usado para requisição.
* **method** - método de requisição para ser usada.
* **baseURL** - pré-adicionada na URL a menos que a url passada seja absoluta.
  + útil para instâncias do axios passar URLs rel.
* **transformRequest** - muda os dados da requisição antes da mesma ser enviada para o servidor.
  + aplicada somente com put, post, patch e delete.
  + última função no array deve retornar uma string ou buffer da instância, ArrayBuffer, FormData ou Stream.
  + permite modificação do cabeçalho.
* **transformResponse** - permite mudança de dados antes de chegarem ao then/catch.
  + semelhante ao anterior.
* **headers** - cabeçalhos customizados para serem enviados.
* **params** - parâmetros de URL que serão enviados junto com a requisição.
  + **OBS: deve sempre ser um objeto ou obj de URLSearchParams**.
* **paramsSerializer** - função opcional que comanda a serialização dos params.
* **data** - dados a serem enviados no corpo da requisição.
  + aplicado em **post, put, delete, patch**.
  + **Nota: se emitido o transforme, o tipo deve ser: sring, objeto, simples, arrayBuffer, arrayBufferView, URLSearchParams**.
  + **No navegador**: FormData, File, Blob.
  + **Node**: Stream e Buffer.
* **timeout** - tempo antes da requisição acabar em milissegundos.
  + se requisição levar tempo maior, será abortada.
* **withCredentials** - indica se as solicitações de controle de acesso entre sites.
* **adapter** - tratamento especializado de solicitações.
  + retorna uma promessa e uma resposta válida.
* **auth** - indica que autenticação básica do HTTP deve ser usada e fornece as credenciais.
  + sobrescreve o cabeçalho de Authorization já existente.
  + **Nota: apenas autenticação HTTP básicas são configuráveis**.
* **responseType** - indica o tipo de dados que o servidor irá responder.
  + **opções**: arrayBuffer, document, json, text, stream.
  + **no navegador**: blob.
* **responseEncoding** - codificação a ser usada para resposta de decotificação.
  + **OBS: ignorado para responseType de stream ou solicitações do lado do cliente**.
* **xsrfCookieName** - nome de cookie para utilizar como um valor para o token xsrf.
* **xsrfHeaderName** - nome do cabeçalho http que carrega o valor do token.
* **onUploadProgress** - permite manusear o progresso do evento de upload.
  + serve apenas no navegador.
* **onDownloadProgress** - permite manusear o progresso do evento para downloads.
* **maxContentLength** - define tamanho máximo do conteúdo da resposta http em bytes permitido.
* **maxBodyLength** - define o tamanho máximo permitido do conteúdo http.
  + apenas para node.
* **validateStatus** - resolve ou rejeita a promessa de um determinado código de status de resposta http.
  + promessa será resolvida se retornar true.
* **maxRedirects** - define o número máximo de redirecionamentos para seguir em node.js.
  + se 0, nenhum redirecionamento é permitido.
* **socketPath** - define o unix socket para ser usado em node.js.
  + apenas socketPath ou proxy podem ser especificado.
* **httpAgent e httpsAgent** - define agente personalizado para ser usado quando perfomar em http ou https, no node.js.
* **proxy** - define o nome do host, porta e o protocolo do proxy do servidor.
  + **http\_proxy**: define o próprio proxy.
  + **https\_proxy**: variáveis de ambiente.
  + **no\_proxy**: lista de domínios que não devem utilizar proxy.
  + **false**: desabilita proxies
  + **auth**: indica que o http basic auth deve ser usado para conectar no proxy e fornecer credenciais.
* **cancelToken** - token de cancelamento usado para cancelar a requisição.
* **decompress** - indica se corpo de resposta deve ou não ser comprimido automaticamente.
  + **Nota: se definido como true irá remover também o cabeçalho content-encoding de todos os objetos resposta**.

**Exemplo**:



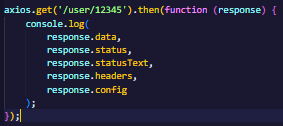




# **8. Esquema de Resposta**

**Informações de uma resposta para requisição**:

* **data** - fornecida pelo servidor.
* **status** - código de status http da resposta do servidor.
* **statusText** - mensagem de status http.
* **headers** - cabeçalhos http que o servidor respondeu.
  + exemplo: response.headers[‘ content-type ’].
* **config** - configuração fornecida ao axios para requisição.
* **request** - requisição que gerou a resposta.
  + última instância da requisição do cliente em node.js.

**Exemplo**:  


# **9. Configurações de Padrões**

Serão aplicadas em todas as requisições.

**Exemplo: configurações globais do axios**.



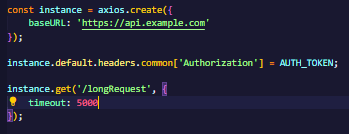
**Exemplo: padrões de instâncias personalizadas**.



**Ordem de precedência**:

1. bibliotecas encontradas em lib/defaults.js (Menor precedência).
2. propriedades da instância.
3. config argumentos de requisição (Maior precedência).

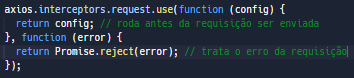
**Exemplo**:



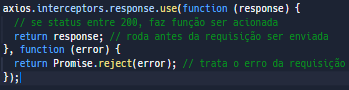
# **10. Interceptadores**

Intercepta respostas antes de serem manipuladas pelo then / catch.

**Exemplo: interceptador de requisições**.



**Exemplo: interceptadores de respostas**.



**eject** - remove um interceptador no código.

* axios.interceptors.request.eject(NameInterceptor).

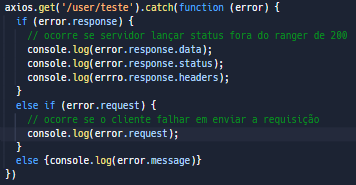
**Nota: interceptadores podem ser aplicados dentro de uma instância do axios**.

* instance.interceptors.request.use(callback).

**Exemplo**:



# **11. Manipuladores de Erros**

**Exemplo**:  


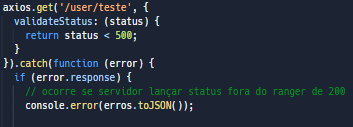
**validateStatus** - define códigos http que devem lançar erros.

**Exemplo**:



**toJSON** - permite obter um objeto com mais informações sobre o erro http.

**Exemplo**:



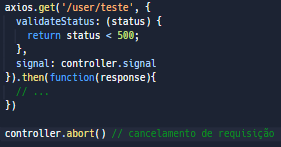
# **12. Cancelamento**

## **12.0 AbortController**

**OBS: disponível somente a partir da v0.22.0**.

Oferece suporte para cancelar requisições em forma de API fetch.

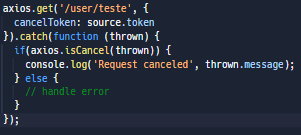
**Exemplo**:



## **12.1 CancelToken (DESCONTINUADO)**

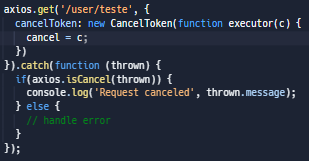
Baseada em cancelable promises proposel.

**Exemplo: criação de token de cancelamento**.





**Exemplo: criação de token de cancelamento com função executora.**



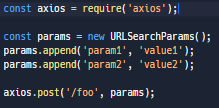
# **13. Corpo de Codificação de URL**

Por padrão os objetos são serializados pela axios para o formato de JSON.

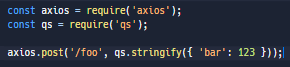
**Formas de enviar os dados em formato urlEncoded**:

1. **navegador** - utiliza-se a API **URLSearchParams** ou a **biblioteca QS**.
   * api não é suportada por todos os navegadores.

**Exemplo**:



**Exemplo: realizando o mesmo processo com a biblioteca qs**.

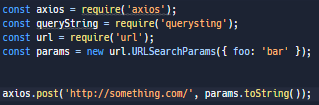


1. **Node.js**.
   * **query string**.

**Exemplo**:



outro modo,



**OBS: a biblioteca qs pode ser utilizada para restringir objetos aninhados, pois queryStrings tem problemas neste caso de uso**.

* + **form data**.

**Exemplo**:  
