E-BOOK
DO ESTUDANTE

Requisições Json (API)

Enviando Cartas

RESILIA

Todos os direitos reservados
©2023 Resilia Educação

SUMÁRIO

Todos os direitos reservados
©2023 Resilia Educação

Requisições Json (API)

Enviando Cartas

Neste *e-book*, vamos compreender o universo de **requisições JSON**, e isso vai nos ajudar a entender como os sites da *web* funcionam e como eles conseguem apresentar toda aquela quantidade de dados e informações.

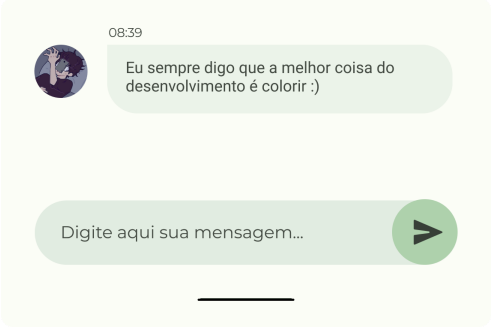
Para começar, deixamos aqui um *spoiler*: os dados que vemos não ficam escritos em HTML. Na verdade, eles ficam armazenados em um banco de dados, e quem faz a extração desses dados e os envia para nossa aplicação é a API (Interface de Programação de Aplicações).

Isso acontece, por exemplo, nas redes sociais que usamos. Elas contêm dados de muitas pessoas, e todos esses dados ficam armazenados em um banco de dados. A API serve para pegar os dados desse banco e os disponibilizar via HTTP (protocolo da internet). Essa comunicação com a API é feita através daquilo que chamamos de **requisição**, e isso significa que, quando falamos em requisições JSON, estamos falando de uma forma de comunicação com a API.

Para entender melhor como os *sites* da *web* funcionam e para conseguir construir um site que também utiliza esses recursos, vamos juntos nesta jornada explorar de forma prática todos esses conceitos que nos foram apresentados.

CONTEXTUALIZANDO

Pense no seguinte exemplo: imagine que nós trabalhamos em uma empresa onde precisamos desenvolver um aplicativo de chat que precisa conter os dados de mensagens trocadas entre duas pessoas. Vamos dar uma olhada em como isso foi esboçado pelo designer dessa empresa:



Todos os direitos reservados
©2023 Resilia Educação

Note que a aplicação que precisaríamos desenvolver teria os seguintes dados: uma foto e uma mensagem. Pensando em como ela funcionaria, teríamos um site (tudo o que vemos visualmente) que, em algum momento, iria precisar se comunicar com uma API para pegar os dados de cada mensagem enviada no chat. Para isso, o site iria realizar uma requisição, ou seja, um tipo de comunicação com uma API para captar os dados. Isso acontece todos os dias, mas muitas vezes nós nem percebemos que um determinado dado vem de uma API — afinal, o que nós vemos são somente os dados.

Porém, essa comunicação é superimportante e é uma das funcionalidades mais utilizadas por qualquer site da web. Montar sistemas que se comunicam é hoje uma rotina do mercado e, por isso, é importante que nós aprendamos a realizar tal funcionalidade.

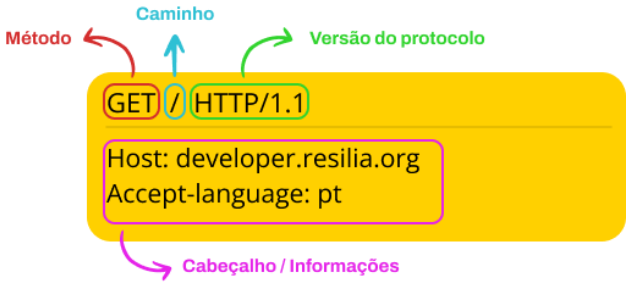
Vamos então entender como funciona uma requisição e como nós conseguimos realizar uma através do Javascript?

REQUISIÇÕES

A requisição é o meio de comunicação entre um site e a API na web. Podemos dizer que a função da requisição é igual à função das cartas que as pessoas trocavam para conseguir se comunicar antigamente, inclusive para pedir ou enviar alguma informação.

Seguindo essa lógica, sempre que um site precisa captar um dado, ele envia uma carta para a API, pedindo uma informação. Assim que recebe a carta, a API responde ao site com outra carta, contendo os dados que foram solicitados.

De forma técnica, é assim que funciona uma requisição:



Em relação aos pontos destacados pelas setas, precisamos saber que:

- O **método** define o tipo de requisição que iremos realizar. O método *GET*, por exemplo, informa que a requisição será do tipo “pegar”, ou seja, o *site* irá se comunicar com uma API para pegar algum dado. Existem outros métodos, como o *POST* (enviar um dado), o *PUT/PATCH* (atualizar um ou vários dados) e o *DELETE* (deletar um ou vários dados).

Todos os direitos reservados
©2023 Resilia Educação

- O **cabeçalho** contém todas as informações da requisição, tais como: local (*host*), linguagem, criptografia, chave de segurança, entre outras.

- O **caminho** define a rota que a requisição irá acessar dentro do *host*. Por exemplo: se o caminho for "*/login*", isso quer dizer que a requisição vai acessar a rota /*login* dentro do *host*, ficando assim: "developer.resilia.org/login".

- A **versão** do http é uma informação para a *web* e para os servidores de como tratar tal requisição.

Essas informações precisam estar presentes em qualquer requisição para ajudar na tarefa de levar os dados ao destino esperado. Isso significa que são essas informações dentro da requisição que vão comunicar aos servidores da *web* por onde ela será transmitida. Assim, elas são importantes para que uma requisição consiga chegar na API e para que depois ela retorne para a nossa aplicação.

Porém, sabemos que uma carta pode não chegar ao seu destino por diversos motivos. Ela pode molhar, rasgar ou até mesmo ser extraviada. Da mesma forma, a requisição pode não fazer a comunicação esperada por diversos motivos, como: pode dar erro de internet, pode ocorrer erro na aplicação, pode não conter os dados necessários, pode não ser da mesma versão do http da API, entre outros erros.

Para nos ajudar a descobrir se houve algum erro na requisição, existem os ***status***. Eles são muito importantes porque nos possibilitam saber se uma requisição teve algum problema ou não. Existem diversos tipos de *status*, mas os mais conhecidos são:

- 100 (Continue)
- 200 (Sucesso)
- 201 (Criado)
- 300 (Escolhas múltiplas)
- 400 (Request inválida)
- 404 (Não encontrado)
- 500 (Erro interno)

Para saber mais sobre os *status*, consulte os seguintes *sites* que abordam o tema de forma descontraída:

HTTP Cats HTTP Dogs

Todos os direitos reservados
©2023 Resilia Educação

Já entendemos, então, como a requisição funciona. Mas ainda não sabemos como os dados são enviados.

Nas cartas, as informações eram escritas em um papel, e o papel era armazenado dentro de um envelope. Nas requisições, nós utilizamos o chamado *body* para o armazenamento dos dados. Em tradução literal, *body*, do inglês, significa *corpo*. Portanto, é o corpo de uma requisição que armazena os nossos dados, e estes dados são armazenados em uma estrutura JSON.

Mas, afinal, o que é o JSON?

De acordo com a documentação MDN, "*JavaScript Object Notation* (JSON) é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto JavaScript", ou seja, o JSON nos permite estruturar nossos dados para que eles possam ser enviados pela *web*.

A sintaxe do JSON se inicia com um **objeto {}** e, dentro desse objeto, há uma sintaxe organizada que contém sempre **uma chave** e **um valor**, por exemplo:

```
{"nome": "jujuba"}
```

Note que há:

- **um objeto**: o que define um objeto é a presença das chaves englobando os dados.
- **uma chave**: a chave é algo único dentro do objeto (não se repete) e ela nos permite acessar o valor. Neste caso, a nossa chave é "nome".
- **um valor:** o valor é definido logo após os **:** (dois-pontos). Neste caso, o nosso valor é "jujuba".

O **JSON** sempre irá manter essa estrutura, então ele sempre terá um objeto, chaves e valores. Isso nos permite estruturar os dados através de chaves (nomes), permitindo, assim, que tais dados possam ser transportados internet afora.

Agora que sabemos como uma requisição funciona, vamos entender como a requisição é realizada dentro do Javascript.

Dentro do Javascript, para realizarmos uma requisição usando o protocolo HTTP, utilizamos o método **XMLHttpRequest**. Observe como é a sintaxe de declaração dele:

```
const xhr = new XMLHttpRequest();c
```

Todos os direitos reservados
©2023 Resilia Educação

Note que criamos primeiramente uma *const* (constante) de nome *xhr* e, logo em seguida, nós atribuímos a ela uma nova requisição HTTP. Após declararmos uma nova requisição, precisamos configurar essa requisição e, para isso, usamos o método *open*:

```
xhr.open("GET", "https://pokeapi.co/api/v2/pokemon/ditto");
```

Com o método *open*, conseguimos abrir a requisição e definir as informações dessa requisição. No exemplo acima, podemos ver o **método** utilizado — GET — e o **caminho** da API que será utilizada para captar os dados dentro do *host*. Após configurarmos as informações necessárias, precisamos enviar a requisição usando o *send*:

```
xhr.send();
```

A propriedade *send* envia a requisição ao *host* declarado, o qual irá nos retornar caso tudo tenha sido realizado com sucesso. Porém, é necessário ainda informar a requisição de que queremos da API uma resposta em JSON. Para isso, utilizamos a propriedade *responseType*:

```
xhr.responseType = "json";
```

Quando definimos que a resposta precisa ser em JSON, é possível observar se houve alguma mudança na variável *xhr*: se por acaso o dado mudar, isso significará que a requisição trouxe alguns dados, sejam eles de sucesso ou de erro.

Para isso, usamos a propriedade *onload* e, dentro dessa propriedade, verificamos o *status*. Se o *status* for igual a 200 (sucesso), nós teremos que usar a propriedade *response* para pegar os dados que nos foram retornados e, logo em seguida, os dados serão exibidos no console da forma como vemos abaixo:

```
xhr.onload = () => {
  if (xhr.status == 200) {
    const data = xhr.response;
    console.log(data);
  } else {
    console.log(`Erro: ${xhr.status}`);
  }
};
```

Todos os direitos reservados
©2023 Resilia Educação

Note que há um *else*. Então, se a requisição não tiver um status 200 (sucesso), nós iremos exibir uma mensagem de erro com o *status* que nos foi enviado para saber qual tipo de erro ocorreu.

Esse tipo de requisição nos permite captar dados de uma API e, consequentemente, criar *sites* que não contenham seus dados principais no HTML, mas sim em uma API, o que é muito **importante para a segurança dos dados**.

Já pensou se todos os dados fossem salvos em HTML? Nossas senhas, nossos *e-mails* e nossas informações pessoais... Com certeza, haveria um caos de vazamentos de senhas e de *e-mails*, e não haveria segurança. Daí a importância do armazenamento em banco de dados e da comunicação com a API, os quais permitem que todos os nossos dados pessoais sejam salvos com a maior segurança possível.

Sabemos que o hábito de colocar em prática tudo aquilo que estudamos contribui para fixarmos melhor o conteúdo estudado e nos possibilita avançar e aprofundar no tema. Então, agora, vamos praticar o que já sabemos sobre requisições.

Atividade: Alternando gatos

Com base nos exemplos vistos no *e-book* até o momento, realize uma requisição para a API Cataas na rota de um gato aleatório. Para isso, utilize o método do tipo GET. Quando finalizar, coloque na página o dado que foi retornado da requisição que você realizou.

Dica: para jogar um elemento na tela, utilize document.write e utilize a *tag* imagem juntamente com o atributo **href** e o *link* retornado pela API.

Para refletir

- O que é uma requisição?
- Do que é composta uma requisição?
- Qual é a importância das requisições?
- Qual é a diferença entre uma carta e uma requisição?
- Como poderíamos deixar nossa requisição segura? Para você, qual é a importância de um dado seguro?

Todos os direitos reservados
©2023 Resilia Educação

AJAX E PROMESSAS

Como sabemos, existem diversas maneiras de executar qualquer tarefa dentro do mundo da programação. No caso das requisições, uma maneira diferente de realizá-las é utilizando o AJAX, que é um método disponibilizado pelo jQuery.

O jQuery é uma biblioteca que nos disponibiliza recursos para o Javascript. Existem diversas outras bibliotecas pela *web*, porém o jQuery é umas das mais usadas no mercado para dar "superpoderes". O jQuery adiciona métodos e funções novas ao Javascript, permitindo, assim, que ele fique mais intuitivo e que tenha mais opções de recursos. É importante conhecê-lo para que possamos utilizar seus diversos recursos em nossos projetos.

Para instalar o jQuery, acesse o *site* principal deles, clique em *download* e selecione o jQuery minimizado (*compressed*) e para produção (*production)*. Um *link* se abrirá com uma página que terá muitos códigos Javascript.

Copie o *link* e o chame dentro de uma *tag script* no atributo *src* para que, quando nosso HTML for aberto, ele carregue o arquivo Javascript do jQuery através do *link* copiado.

Se preferir, você pode baixar o arquivo através do *link* que abrir e anexar o arquivo ao invés do *link* da *tag script*.

```
 <body>
    <script
src="https://code.jquery.com/jquery-3.6.3.min.js"></script>
  </body>
```

Após carregado o arquivo *script* do jQuery, ele nos permitirá utilizar alguns recursos, entre eles o AJAX.

Utilizando o AJAX, podemos criar requisições de forma totalmente intuitiva e mais organizada visualmente, como mostra o exemplo abaixo:

```
$.ajax({
  method: "GET",
  url: "https://pokeapi.co/api/v2/pokemon/ditto",
  success: function (result) {
    console.log(result);
  },
  error: function (erro) {
    console.log(erro);
  },
});
```

Todos os direitos reservados
©2023 Resilia Educação

Note que iniciamos com um sinal de **$** (cifrão). Esse sinal é uma chamada direta a todos os recursos diretos do jQuery, mostrando que estamos utilizando o método AJAX dentro do jQuery.

Como mostra o exemplo acima, o método AJAX recebe um argumento, que é o objeto, e este objeto contém todas as informações da requisição, ou seja, o método, o caminho (URL), entre outras possíveis informações.

Note também que estamos criando métodos (*método* é o nome que define uma função dentro de um objeto). O método de *success* (sucesso) e o método de *error* (erro) são criados pelo AJAX para executar uma ação caso a requisição seja feita com sucesso ou com erro, respectivamente.

Isso significa que, ao utilizar o AJAX, não precisamos mais verificar e validar os *status*, pois o próprio AJAX realiza essa verificação. Dessa forma, temos duas vantagens com o AJAX: a quantidade do nosso código diminui, tornando-o mais legível, e não precisamos criar verificação para todos os possíveis *status*. Outra vantagem é que o AJAX nos permite configurar e realizar uma requisição de uma forma bastante estruturada usando um objeto de configuração.

Vimos até o momento dois modos bastante diferentes de realizar uma requisição, porém algo bastante comum no desenvolvimento *web* é que existem vários meios para alcançar um objetivo. Então agora vamos explorar mais uma forma diferente que podemos utilizar para realizar uma requisição: o *Promises* (Promessas).

Quando falamos em *promessas*, estamos falando em algo que usamos diariamente na *web*. Quando enviamos uma mensagem para alguém, por exemplo, sabemos que essa mensagem não tem horário para chegar ao seu destino e que também não há nenhuma garantia de que ela chegará de fato ao destino, pois pode ocorrer algum erro.

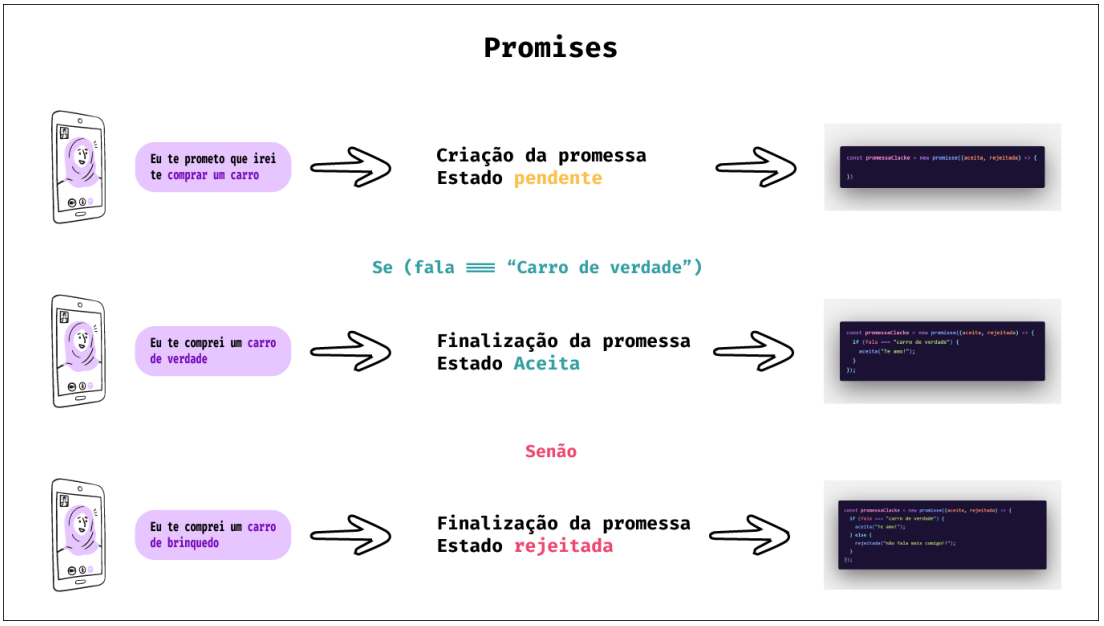
O que acontece é que o sistema nos promete que ele vai tentar enviar a mensagem o mais rápido possível, e isso acontece em todos os casos de requisições. Nós nunca saberemos quanto tempo levará uma requisição devido aos inúmeros fatores possíveis, como erros, falta de internet, entre outros.

Todos os direitos reservados
©2023 Resilia Educação

Uma promessa tem três estados:

- O estado **pendente** - Ocorre quando algo foi prometido, mas não foi concluído.
- O estado de **resolvida** - Ocorre quando a promessa é realizada e concluída com sucesso, por exemplo quando eu prometo algo a alguém e cumpro exatamente essa promessa.
- O estado de **rejeitada** - Ocorre quando a promessa é realizada e concluída com rejeição, por exemplo quando eu prometo comprar algo para alguém, mas compro algo diferente do que foi prometido (de forma negativa).

Para entendermos melhor, vamos ver um exemplo visual de uma promessa:



No primeiro exemplo, vemos a criação da promessa com o estado pendente, e isso ocorre quando declaramos nossa promessa dentro do Javascript, desta forma:

```
let promessa = new Promise((aceita, rejeitada) => {});
```

Note que, neste código, existe uma variável de nome “promessa” que está recebendo uma nova promessa. Dentro da promessa, existe uma função de seta *(a função de seta é uma função mais curta para digitar e também se comporta como uma função anônima)*, e esta função está recebendo dois parâmetros, um de nome “aceita” e outro de nome “rejeitada”. Dessa forma, temos a nossa promessa já criada, então agora precisamos começar a validar e determinar se a promessa será aceita ou rejeitada.

Todos os direitos reservados
©2023 Resilia Educação

Neste segundo exemplo abaixo, vemos um exemplo de verificação com a seguinte condição: “se fala for estritamente igual a ‘carro de verdade’”. Ou seja, para definirmos um estado da promessa, precisamos criar as condições para saber se ela foi aceita, como em:

```
let promessa = new Promise((aceita, rejeitada) => {
  if(fala === "Carro de verdade"){
    aceita("Te amo!")
  }
});
```

Note que estamos usando o *if* para verificar *se fala é estritamente igual a “Carro de verdade”*. Se de fato for igual, finalizaremos a promessa com o estado **aceita** e enviaremos a resposta “Te amo!”.

Mas e se “fala” não for estritamente igual a “Carro de verdade”?

Então iremos rejeitar a promessa, com o estado **rejeitada**, usando o *else*, como podemos ver no exemplo abaixo:

```
let promessa = new Promise((aceita, rejeitada) => {
  if(fala === "Carro de verdade"){
    aceita("Te amo!")
  } else {
    rejeitada("Não fala mais comigo!")
  }
});
```

Note que uma mensagem será enviada não só quando a promessa for aceita, mas também quando ela for rejeitada, e isso ocorre para que possamos receber esse *feedback* da promessa.

Para pegar esse dado passado nos estados *aceita* e *rejeitada*, utilizamos os métodos *then* e *catch*. O método *then* permite esperar um sucesso da promessa, já o método *catch* permite esperar um erro da promessa. Então, para pegarmos a mensagem da promessa aceita, utilizamos o *then*. Para pegarmos a mensagem da promessa rejeitada, utilizamos o *catch*, como mostra o exemplo a seguir:

Todos os direitos reservados
©2023 Resilia Educação

```
let promessa = new Promise((aceita, rejeitada) => {
  if (fala === "Carro de verdade") {
    aceita("Te amo!");
  } else {
    rejeitada("Não fala mais comigo!");
  }
})
  .then((mensagem) => {
    console.log(mensagem);
  })
  .catch((mensagem) => {
    console.log(mensagem);
  });
```

Uma promessa nos permite executar requisições de forma assíncrona, isto é, podemos esperar a requisição ser finalizada (com sucesso ou erro). Um exemplo disso dentro do Javascript é o método *fetch*, que nos permite realizar uma promessa de requisição a uma API de forma bastante simples, diferentemente das outras possibilidades que estudamos anteriormente. Veja o exemplo abaixo:

```
fetch("https://pokeapi.co/api/v2/pokemon/ditto")
  .then((dadosEmTexto) => dadosEmTexto.json())
  .then((dadosEmJson) => console.log(dadosEmJson))
  .catch((erro) => {
    console.log(erro);
  });
```

Note que o método utilizado é o *fetch* e que ele está recebendo a URL de uma API, onde ele irá fazer a requisição do tipo GET.

Você percebeu que há dois métodos *then* no exemplo acima?

Isso acontece porque, quando recebemos os dados da requisição, eles vêm em formato de texto, pois toda aquela estrutura JSON tem que ser transformada em texto para que seja possível sua passagem pela internet. Então, quando esses dados chegam para nós, eles chegam em formato de texto.

Nesse sentido, o primeiro *then* serve para pegar os dados vindos da requisição e realizar a conversão de texto para JSON. O método JSON, que já estudamos, é uma promessa também, o que nos permite esperar pela finalização.

Todos os direitos reservados
©2023 Resilia Educação

Apenas quando o método JSON finaliza a conversão dos dados é que entramos no segundo *then*. Mostramos esses dados, agora convertidos em JSON, no console. Se ocorrer algum erro na requisição, o dado irá diretamente para o *catch*, e isso será mostrado no console.

Atividade: Obtendo tronos

Usando somente o *fetch*, realize uma requisição para a Game Of Thrones Quotes API na rota onde conseguimos pegar uma mensagem aleatória. Para isso, utilize o tipo de requisição GET. Em seguida, considerando o dado que for retornado da requisição, transforme-o em JSON e coloque-o no console ou na tela.

Como explicado neste *e-book*, o JSON é amplamente utilizado no mercado da *web* e em todos os lugares onde podemos interagir com dados. Portanto, é extremamente importante aprender sobre ele, além de ser extremamente valioso para o nosso desenvolvimento, pois nos permite entender como cada percurso das requisições funciona e como cada etapa é importante para que no final o usuário consiga receber seus dados.

Este conteúdo é bastante complexo, então precisamos ter o hábito de colocá-lo em prática e de sempre nos atualizar sobre ele.

Para refletir

- Qual é a utilidade do fetch? Por que usar o fetch e não o XMLHttpRequest?
- O que é uma promessa?
- Quantos estados tem uma promessa? Quais estados são esses?
- Quais foram as promessas em sites e aplicativos que você fez hoje?

Todos os direitos reservados
©2023 Resilia Educação

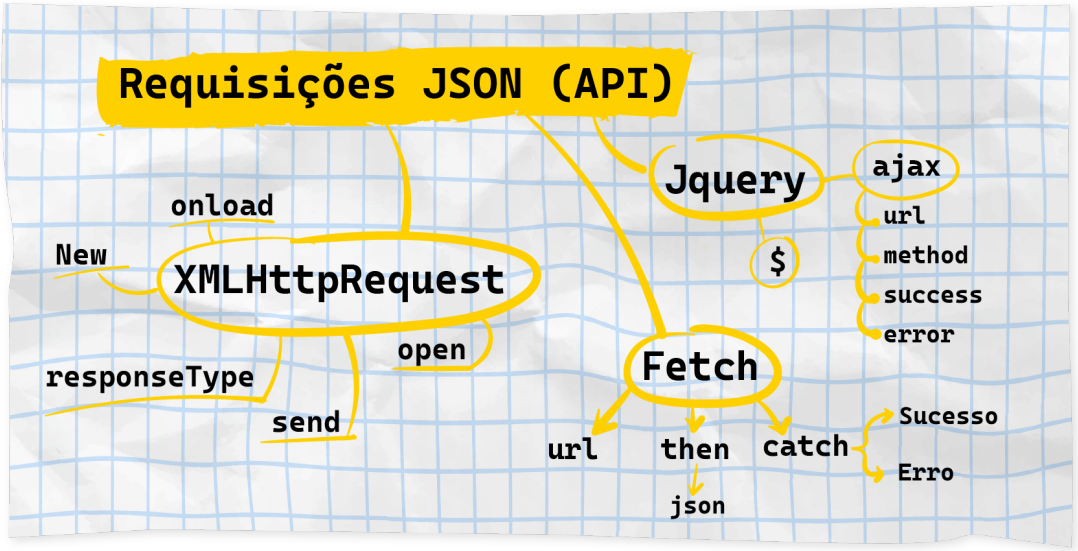
Para ir além

- Como vimos, existem diversas maneiras de se realizar uma tarefa na programação *web* e é importante conhecer todas elas. Para se aprofundar em requisições usando o XMLHttpRequest, consulte o guia a seguir: >https://kinsta.com/knowledgebase/javascript-http-request/<

- Agora que sabemos a importância das requisições usando jQuery e AJAX e a popularidade do jQuery no mercado da *web*, precisamos nos aprofundar no assunto. Para saber mais, acesse: >https://api.jquery.com/Jquery.ajax/<

- O *fetch* é muito usado no Javascript, sendo um marco de simplicidade e agilidade em requisições dentro do Javascript de forma nativa da linguagem. Conheça esta página para aprofundar no assunto: >https://edrodrigues.com.br/blog/primeiros-passos-com-a-api-javascript-fetch/<

NUVEM DE PALAVRAS



Todos os direitos reservados
©2023 Resilia Educação

Até a próxima e
#confianoprocesso

RESILIA

Todos os direitos reservados
©2023 Resilia Educação