

JavaScript

Aula 3

Um pouco mais sobre eventos

Excluindo elementos do DOM (exe1.html)

Os elementos possuem um método *remove()* que permitem sua exclusão do DOM. Ele deve ser invocado no elemento que será excluído.

```
var div = document.createElement("div");  
div.addEventListener('dblclick',function(){  
    this.remove();  
});
```

Excluindo elementos do DOM (exe1.html)

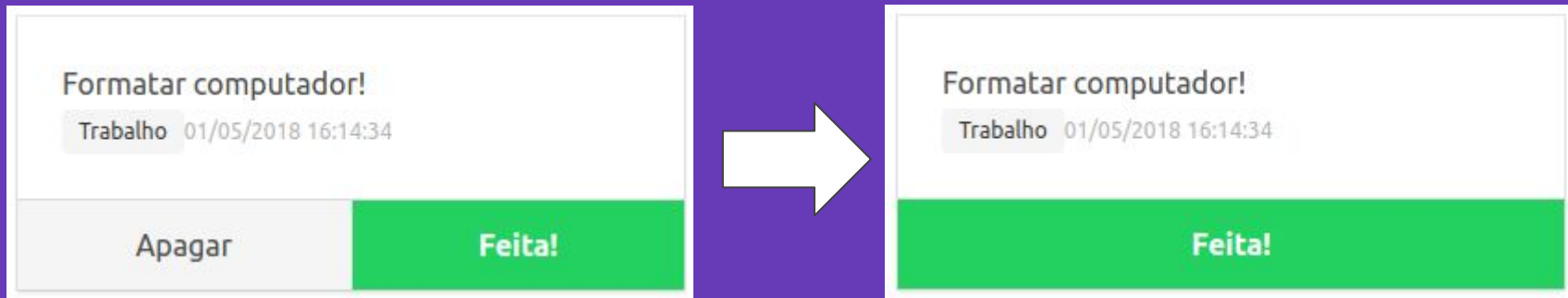
Outra opção é utilizar o método *removeChild()* a partir do pai para remover um dos elementos filhos.

```
var lista = main.querySelectorAll("div");  
if (lista.length > 0) {  
    main.removeChild(lista[0]);  
}
```

Exercício (exe2/index.html)

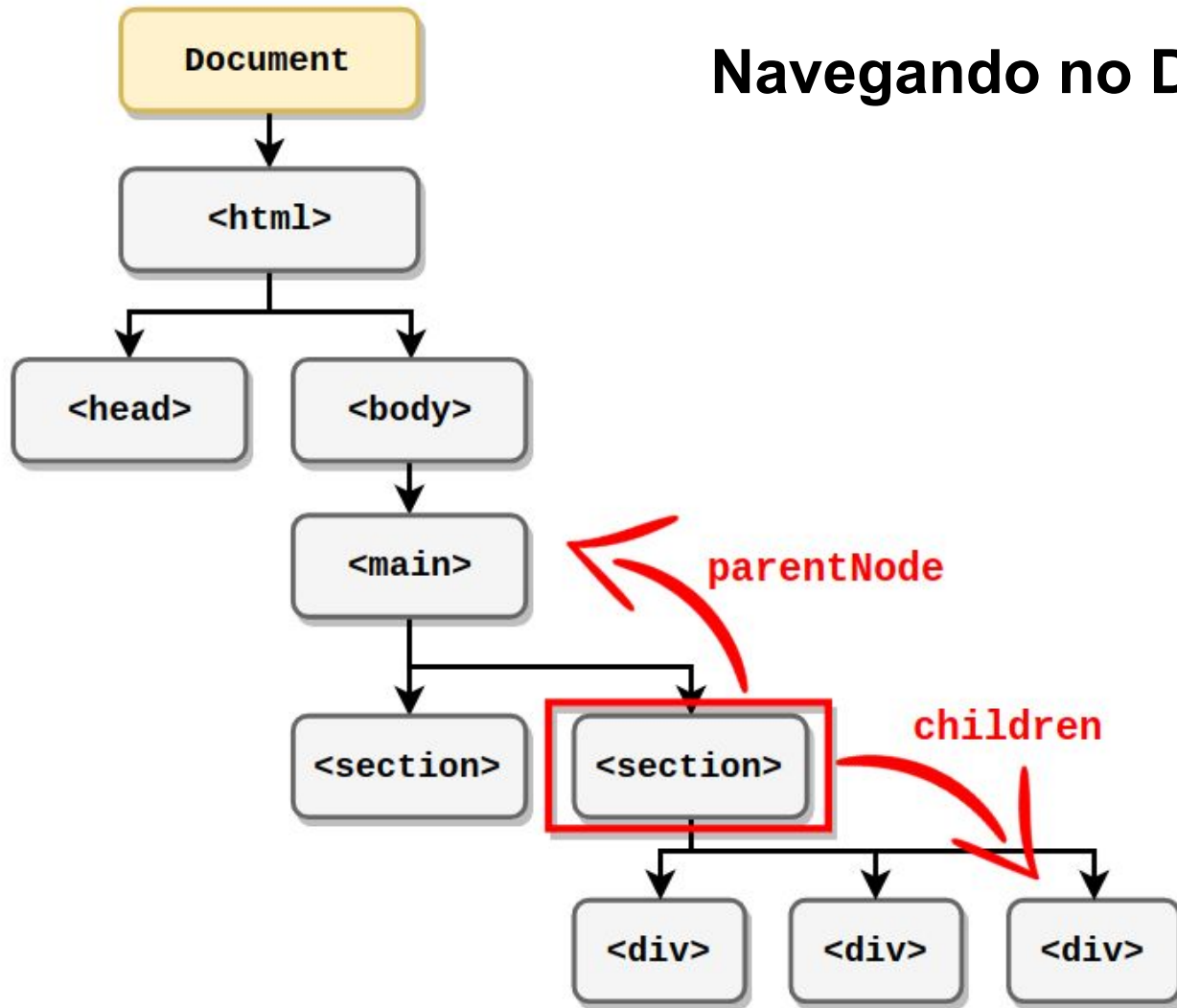
Continuando o projeto TO-DO App, configure um evento de clique no **botão de apagar** das tarefas, execute o método *remove()* para o *this* do evento e veja o que acontece.

Exercício (exe2/index.html)



Por que isso aconteceu?

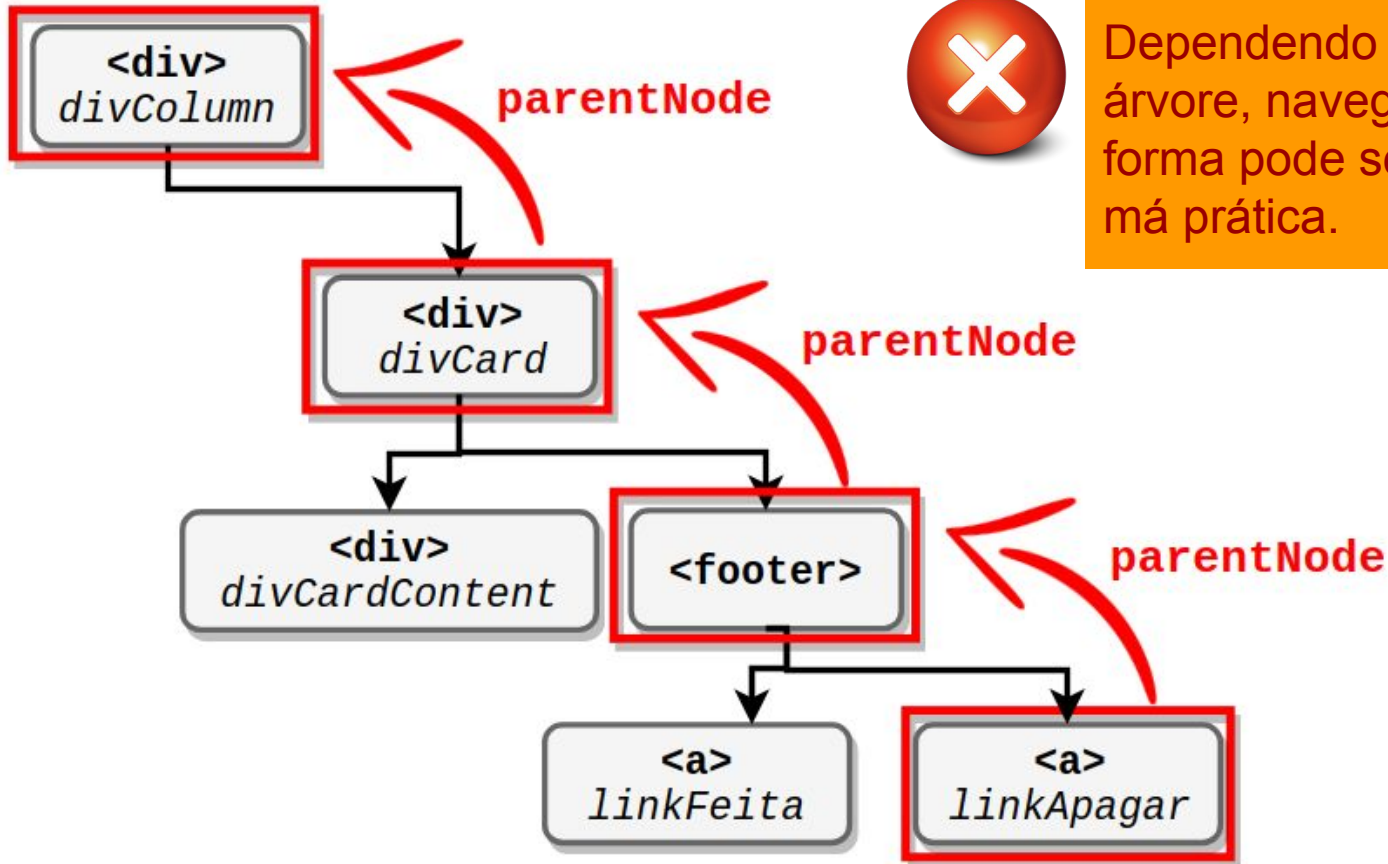
Navegando no DOM (exe3.html)



Exercício (exe4/index.html)

Vamos agora ajustar a TO-DO App para, ao invés de apagar o botão, apagar a tarefa. Neste caso, a partir do clique no botão, percorra o DOM em busca do elemento associado à tarefa.

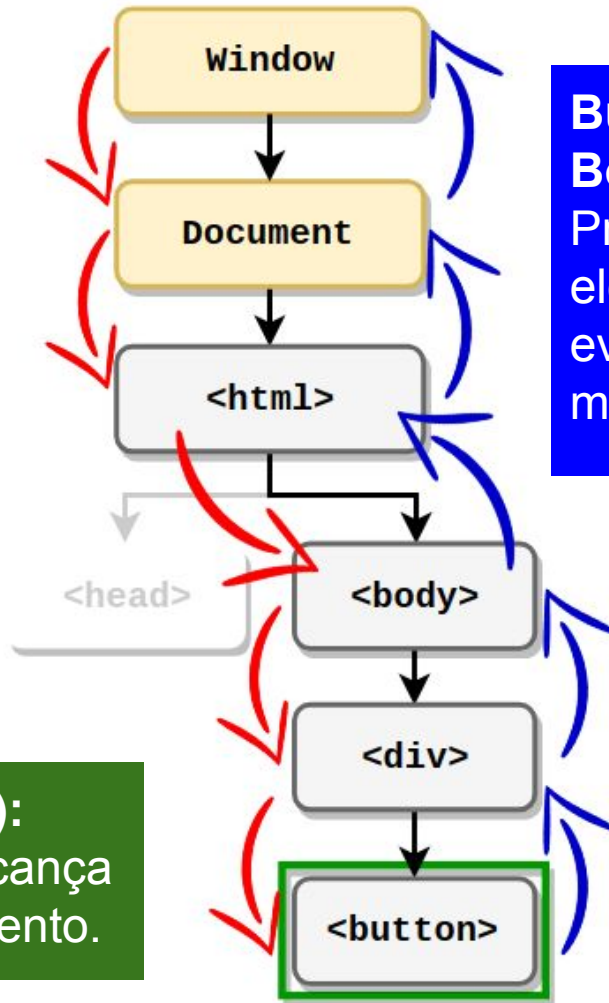
Excluindo tarefas (exe4.html)



Fases de propagação dos eventos no DOM

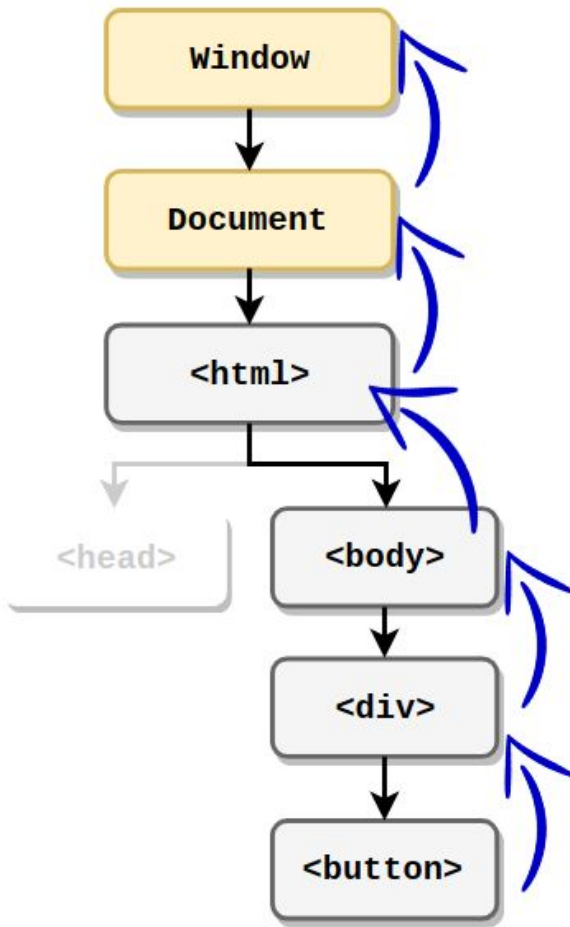
Capturing Phase (Fase de Captura): O evento se propaga do elemento mais alto na hierarquia até o elemento que disparou o evento.

Target Phase (Fase do Alvo): Momento em que o evento alcança o elemento que disparou o evento.



Bubbling Phase (Fase Borbulhante): Propagação a partir do elemento que disparou o evento até o elemento mais alto na hierarquia.

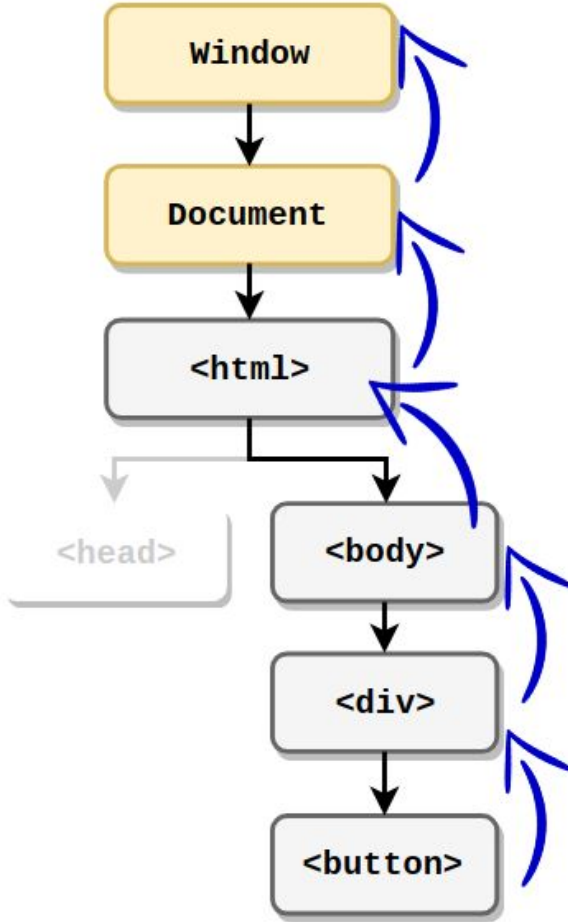
Ok, mas para que serve isso? (exe5.html)



Na prática, a **Bubbling Phase** é útil pois através dela podemos capturar o evento disparado a partir dos elementos mais altos na hierarquia. Por exemplo, suponha um caso em que você quer concentrar o tratamento dos eventos no <body>.

```
var body = document.querySelector('body');
var botaoOk = document.querySelector('#botaoOk');
body.addEventListener('click',function(evento){
    if (evento.target == botaoOk) {
        console.log('Botão OK clidado!');
    }
});
```

Descobrimos qual elemento foi clicado (exe5.html)



```
var body = document.querySelector('body');  
var botaoOk = document.querySelector('#botaoOk');  
body.addEventListener('click',function(evento){  
    if (evento.target == botaoOk) {  
        console.log('Botão OK clidado!');  
    }  
});
```

Note que neste caso é necessário comparar o **alvo** do evento (*evento.target*) com o elemento que deseja controlar pois a tag <body> vai receber eventos de todas suas tags filhas.

Exercício (exe6/index.html)

Altere o projeto TO-DO App para, ao invés de usar as propriedades *parentNone*, realizar os seguintes passos:

- Capturar o evento de clique no elemento associado à tarefa
- Verificar se o clique foi realizado no botão “Apagar”
- Apagar a tarefa

Regras de Negócio no JavaScript

Pontos de experiência no TO-DO App

Vamos agora implementar uma pequena regra para alterar os pontos de experiência do usuário de acordo com as seguintes ações:

- Incluir Tarefa: +1 ponto
- Realizar Tarefa: +1 ponto
- Apagar Tarefa: -2 pontos

✓ TO-DO App

XP

5

lgapontes

Logar

☒ Trabalho ☐ Lazer

Digite sua tarefa

Exercício (exe6/index.html)




Altere o projeto TO-DO App de acordo com os itens abaixo:

- Coloque um ID no `` que exibe o total de experiência do usuário e inicie (no HTML) seu valor como zero.
- Ao incluir uma nova tarefa, obtenha o valor da experiência e acrescente 1 (atualizando o HTML no final).
- Ao apagar uma tarefa, reduza 2 pontos da experiência.
- Ao concluir uma tarefa, acrescente 1 ponto à experiência. Para isso, capture o evento click no botão “Feita”.

Dica: crie uma função para realizar a atualização do `` experiência.

Cores da experiência no TO-DO App

Vamos agora adicionar classes de cores específicas de acordo com a pontuação do usuário.

Para valores menores que zero, acrescente a classe: <code>is-danger</code>	Para valores entre zero e 4 (inclusive), acrescente a classe: <code>is-warning</code>	Para valores maiores que 4, acrescente a classe: <code>is-success</code>
		

Observação: para que esta lógica funcione, você sempre deverá excluir as classes supracitadas para evitar que a tag acumule classes de estilo indefinidamente.

Exercício (exe6/index.html)

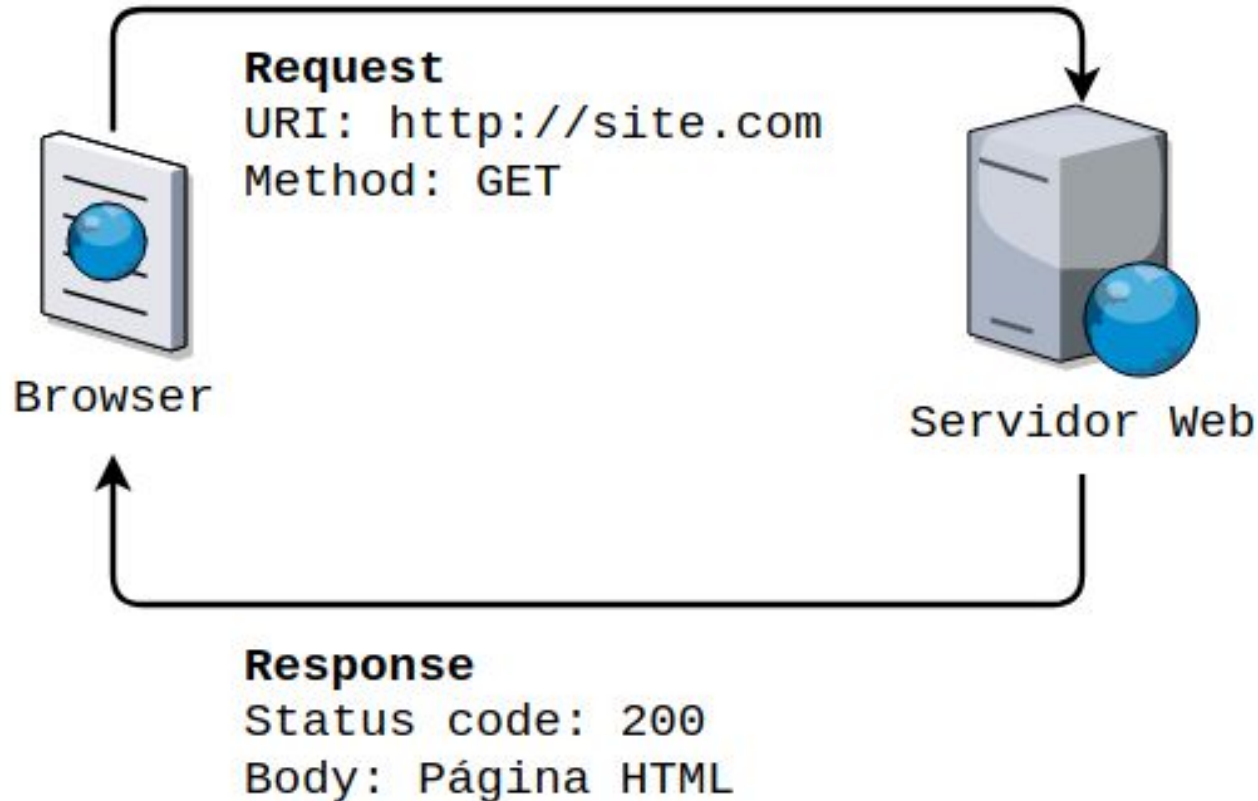
Altere o projeto TO-DO App de acordo com a regra citada no slide anterior, exibindo as cores adequadas de acordo com a pontuação do usuário.

Nunca duplique código de regra de negócio! Se for necessário, crie funções para organizar seu código.

Trabalhando com API

(Application Programming Interface)

Comunicação entre o Browser e o Servidor Web



Google x

← → ↻ Seguro | <https://www.google.com> ☆ ⋮

Responsive ▾ 408 x 74 100% ▾ Online ▾ ⌵

Google







⋮

🔍 📄 | Elements Console Sources **Network** Performance Memory » ⋮ ✕

🔴 🚫 📺 🔍 | View: 📄 📄 ☐ Group by frame ☐ Preserve log ☐ Disable cache ☐ Offline

Filter ☐ Hide data URLs

All | XHR JS CSS Img Media Font Doc WS Manifest Other

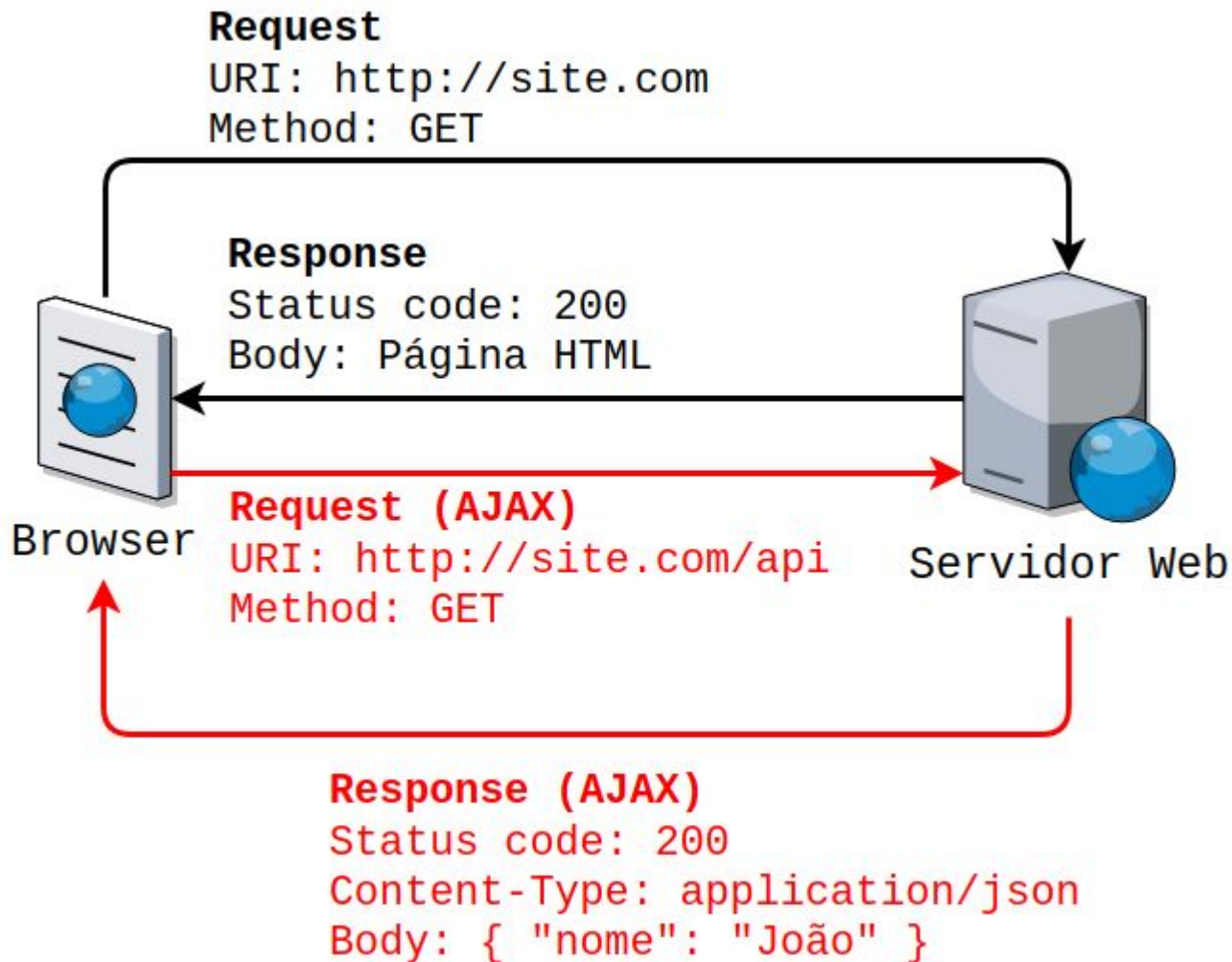
Name	Status	Type	Initiator	Size	Time	Waterfall	4.0 s	6.0 s
 www.google.com	200	doc...	Other	35...	1.4...			
 googlelogo_color_...	200	png	(index)	7.0 ...	28...			
 ai2_00ed8ca1.png	200	png	(index)	3.3 ...	24...			

22 requests | 285 KB transferred | Finish: 7.06 s | DOMContentLoaded: 1.82 s | Load: 6.16 s

AJAX

Asynchronous
Javascript and XML

1. Um evento é disparado no browser
2. O browser envia uma requisição ao servidor
3. O servidor processa e envia uma resposta ao browser
4. O browser trata a resposta para atualizar a página



Content-Type mais utilizados

XML (*Extensible Markup Language*)

```
<xmlcep>
  <cep>27933-140</cep>
  <logradouro>Alameda Raimundo
  Corrêa</logradouro>
  <complemento/>
  <bairro>Glória</bairro>
  <localidade>Macaé</localidade>
  <uf>RJ</uf>
  <unidade/>
  <ibge>3302403</ibge>
  <gia/>
</xmlcep>
```

JSON (*JavaScript Object Notation*)

```
{
  "cep": "27933-140",
  "logradouro": "Alameda Raimundo Corrêa",
  "complemento": "",
  "bairro": "Glória",
  "localidade": "Macaé",
  "uf": "RJ",
  "unidade": "",
  "ibge": "3302403",
  "gia": ""
}
```

Objeto *XMLHttpRequest*()

```
var xhr = new XMLHttpRequest();  
xhr.open('GET','http://sitelegal.com/api');  
  
xhr.addEventListener('load',function(){  
    console.log(xhr.responseText);  
});  
  
xhr.send();
```


JavaScript x

file:///opt/repositories/bitbucket/aulas/desenvolvimento-web/aula-js3/exe8.html

Responsive ▾ 423 x 46 100% ▾ Online ▾

27933140 Buscar CEP

Elements Console Sources **Network** Performance Memory >>

View: [Icons] Group by frame Preserve log Disable cache Offline

Filter Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Status	Type	Initiator	Size	...	Waterfall	0.00 s	▲
exe8.html	Finished	doc...	Other	0 B	...			
json/	200	xhr	exe8.htm...	(fro...	...			

2 requests | 0 B transferred | Finish: 12.10 s | DOMContentLoaded: 22 ms | Load: 23 ms

Consulta de CEP via XML (exe7.html)

<https://viacep.com.br/ws/27933140/xml/>

Ao consumir a URI acima, será retornado um XML. Para fazer o parse do XML via JavaScript, deve-se utilizar o objeto *DOMParser()*.

```
var xml = xhr.responseText;  
parser = new DOMParser();  
xmlDoc = parser.parseFromString(xml, "text/xml");  
var logradouroTag = xmlDoc.querySelector("logradouro");  
console.log(logradouroTag.textContent);
```

Consulta de CEP via JSON (exe8.html)

<https://viacep.com.br/ws/27933140/json/>

Ao consumir a URI acima, será retornado um JSON. Para transformá-lo em um objeto JavaScript, deve-se utilizar o método *JSON.parse()*.

```
var json = xhr.responseText;  
var objeto = JSON.parse(xhr.responseText);  
console.log(objeto.logradouro);
```

Caso seja necessário transformar o objeto JavaScript em JSON, deve-se utilizar o método *JSON.stringify()*.

Atenção!

O JavaScript é uma linguagem de **I/O não bloqueante**. Na prática, isso significa que qualquer operação de entrada e saída é assíncrona, ou seja, **NÃO** bloqueia o fluxo de execução do navegador.

Vejamos o exe9.html

I/O não bloqueante (exe9.html)

Isso acontece justamente porque o JavaScript **NÃO** espera a conclusão das chamadas assíncronas para prosseguir com o fluxo de execução. Para resolver isso, precisamos utilizar um famoso recurso conhecido como:

callback

Função callback (exe10.html)

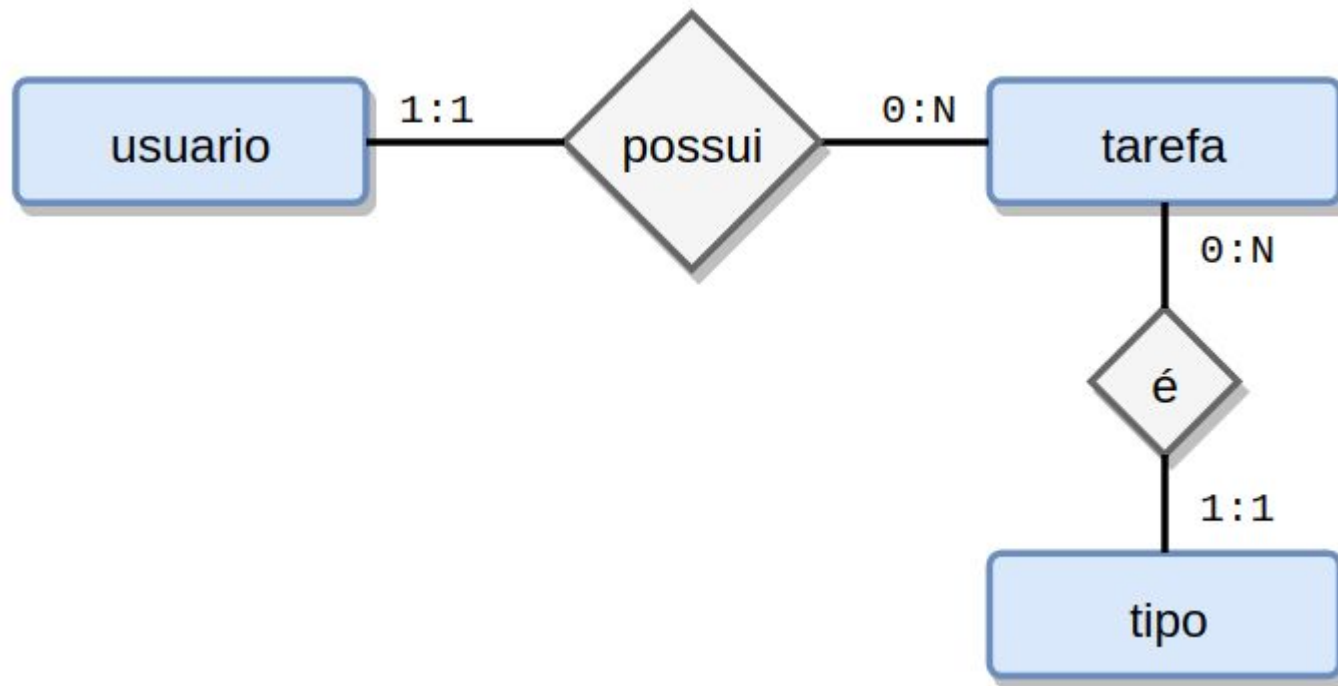
Callback é uma função passada como parâmetro para outra função com o objetivo de ser invocada somente quando a chamada assíncrona terminar.

```
function obterLogradouro(cep, callback) {  
    var xhr = new XMLHttpRequest();  
    xhr.open('GET', 'https://viacep.com.br/ws/' + cep + '/json/');  
    xhr.addEventListener('load', function(){  
        var objeto = JSON.parse(xhr.responseText);  
        callback(objeto.logradouro);  
    });  
    xhr.send();  
}
```

TO-DO App com API

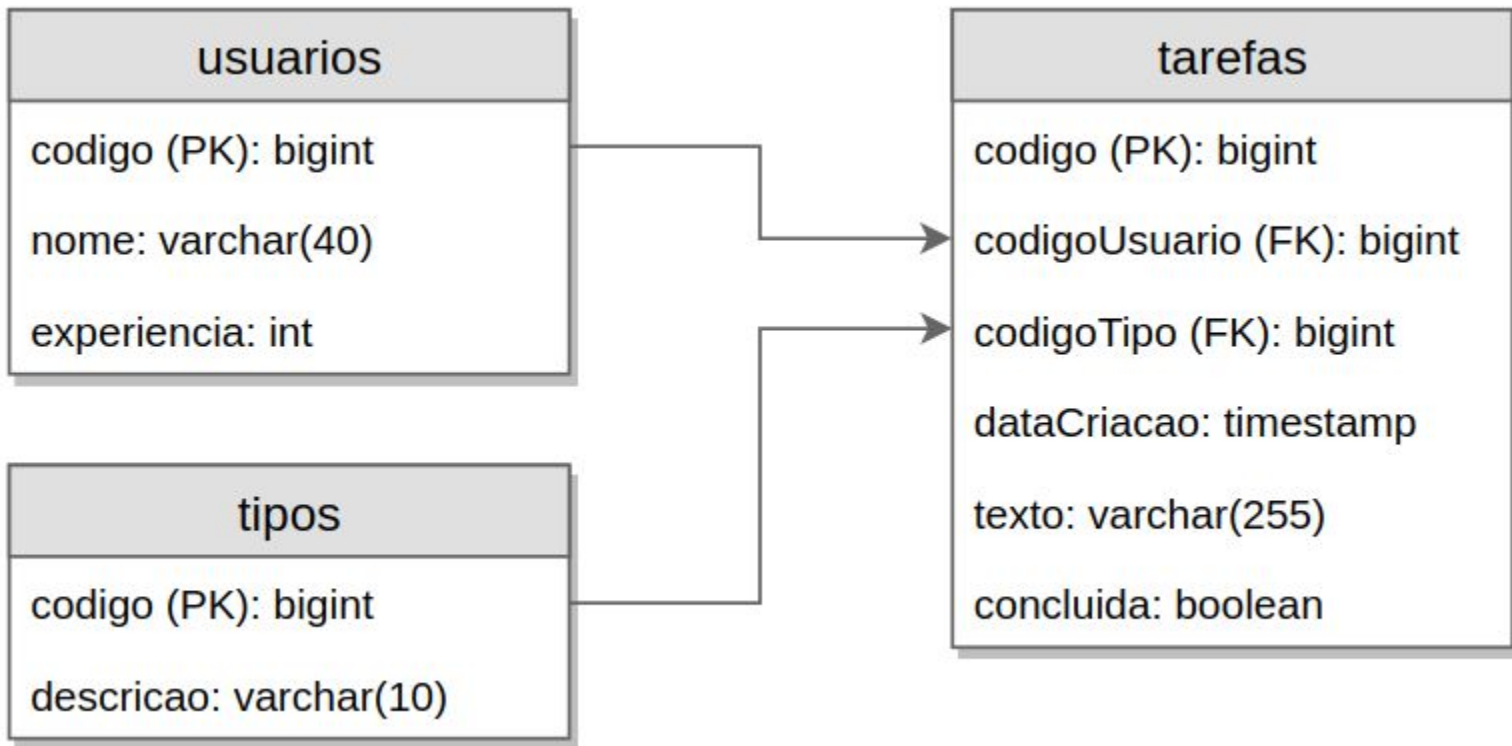
TO-DO App

Diagrama de Entidade e Relacionamento



TO-DO App

Diagrama de Estrutura de Dados



TO-DO App

API para Listar Tipos

URI: <http://lgapontes.com/aulas/todoapp/api/tipos>

Method: GET

Retorno:

```
[  
  {"codigo":"1","descricao":"Trabalho"},  
  {"codigo":"2","descricao":"Lazer"}  
]
```

Veja: exe11.html



Insomnia



POSTMAN



SoapUI



Firefox



Fiddler

HTTP Debugging Proxy

Exercício (exe11.html)

Passos:

1. Cria uma página com um `<button>` e uma ``
2. Ao clicar no `<button>` leia a API
<http://lgapontes.com/aulas/todoapp/api/tipos>
3. Crie `` para cada valor retornado

Dicas: Lembre-se de converter o *xhr.responseText* com *JSON.parse()*

Extra: Coloque uma imagem de loading enquanto a tela aguarda o retorno da API.

Código: 1**Descrição:** Trabalho**Código: 2****Descrição:** Lazer

Exercício (exe12/index.html)



Este exercício vai gerar um pequeno erro na lógica!

Altere o projeto TO-DO App para:

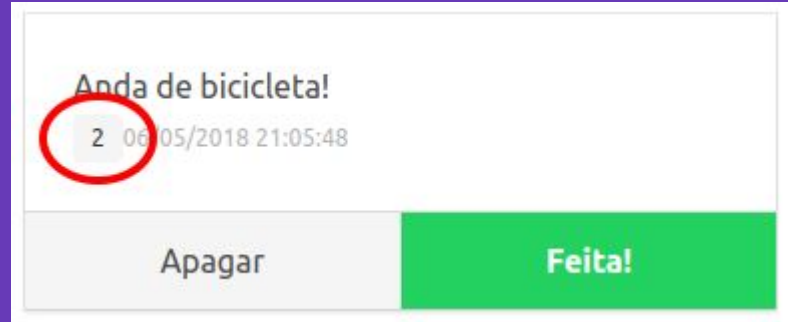
- Obter a lista de tipos da API. Preferencialmente, crie outro arquivo JS para criar as funções de consumo da API.
- Para cada tipo obtido, acrescente no formulário de tarefas um `<input>` do tipo *radio* (com sua `<label>` e texto) conforme abaixo.

```
<label class="radio">  
  <input type="radio" name="tipo" value="Trabalho" checked >  
  <span>Trabalho</span>  
</label>
```

Atenção: o campo *value* deve receber o código do tipo. Para o tipo de código 1, acrescente o atributo *checked*.

Exercício (exe12/index.html)

Veja que após criar tarefas com o novo formulário, o tipo da tarefa aparece com o código ao invés da descrição.

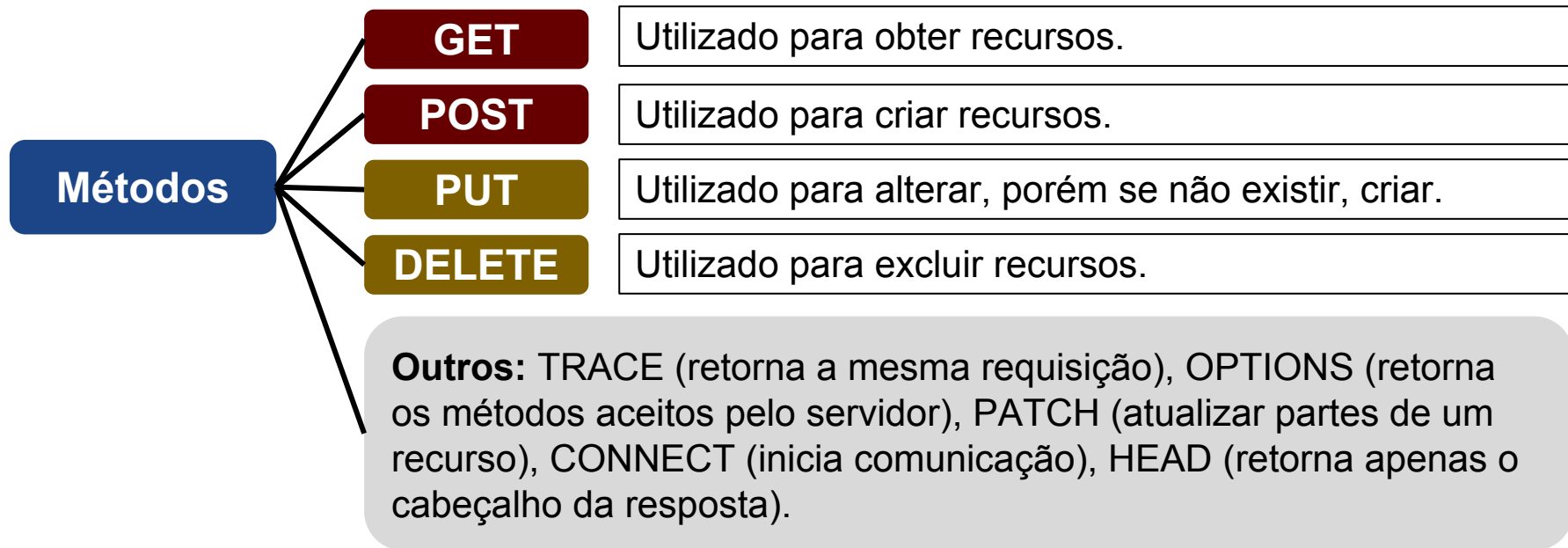


Altere o trecho que obtém o *value* do `<input>` do tipo *radio* para pegar o elemento pai e, a partir dele, executar uma *querySelector()* em busca do `` que guarda o texto da `<label>`.

```
tagRadios.forEach(function(entry){  
  if (entry.checked) {  
    tipo = entry.parentNode.querySelector('span').textContent;  
  }  
});
```

Métodos HTTP e Códigos de Status

Principais métodos HTTP



▼ General

Request URL: `http://lgapontes.com/aulas/todoapp/api/tipos`

Request Method: GET

Status Code: 🟢 200 OK

Remote Address: 192.185.213.160:80

Referrer Policy: no-referrer-when-downgrade

Principais métodos HTTP

Idempotência

Métodos que podem ser chamados várias vezes sem causar problemas no servidor.

Quais?

GET, OPTIONS, HEAD, PUT, TRACE, CONNECT e DELETE

Exemplo:

<http://lgapontes.com/aulas/todoapp/api/tipos>

TO-DO App

API para Listar Usuários

URI: lgapontes.com/aulas/todoapp/api/usuarios

Method: GET

GET lgapontes.com/aulas/todoapp/api/usuarios

Send

200 OK

TIME 271 ms

SIZE 148 B

Body

Auth

Query

Header

Docs



Select a body type from above

Preview

Header 5

Cookie

Timeline

```
1 ▼ [  
2 ▼  {  
3     "codigo": "1",  
4     "nome": "lgapontes",  
5     "experiencia": "3"  
6  },  
7 ▼  {  
8     "codigo": "2",  
9     "nome": "Laura",  
10    "experiencia": "0"
```

\$.store.books[*].author

TO-DO App

API para Login

URI: lgapontes.com/aulas/todoapp/api/login

Method: POST

```
Body: {  
  "nome": "lgapontes"  
}
```

POST localhost/todoapp/api/login

Send

200 OK

TIME 1.47 s

SIZE 59 B

JSON

Auth

Query

Header 1

Docs

```
1 {  
2   "nome": "Luke"  
3 }
```

Preview

Header 4

Cookie

Timeline

```
1 {  
2   "codigo": "3",  
3   "nome": "Luke",  
4   "experiencia": "0",  
5   "tarefas": []  
6 }
```

TO-DO App

API para Listar Tarefas

URI: lgapontes.com/aulas/todoapp/api/tarefas

Method: GET

GET localhost/todoapp/api/tarefas

Send

200 OK

TIME 1.05 s

SIZE 299 B

Body

Auth

Query

Header

Docs



Select a body type from above

Preview

Header 5

Cookie

Timeline

```
1 [
2   {
3     "codigo": "1",
4     "codigoUsuario": "1",
5     "codigoTipo": "2",
6     "dataCriacao": "2018-05-02 23:16:08",
7     "texto": "Navegar na Internet!",
8     "concluida": "1",
9     "tipo": "Lazer"
10  },
```

\$.store.books[*].author

TO-DO App

API para Incluir Tarefa

URI: lgapontes.com/aulas/todoapp/api/tarefa

Method: POST

```
Body: {  
  "codigoUsuario": 3,  
  "codigoTipo": 2,  
  "texto": "Navegar na Internet!"  
}
```

POST localhost/todoapp/api/tarefa

Send

200 OK

TIME 1.66 s

SIZE 202 B

JSON Auth Query Header 1 Docs

```
1 {  
2   "codigoUsuario": 1,  
3   "codigoTipo": 1,  
4   "texto": "Formatar micro!"  
5 }
```

Preview Header 5 Cookie Timeline

```
1 {  
2   "usuario": {  
3     "codigo": "1",  
4     "nome": "lgapontes",  
5     "experiencia": "2"  
6   },  
7   "tarefa": {  
8     "codigo": "4",  
9     "codigoUsuario": "1",  
10    "codigoTipo": "1",
```

TO-DO App

API para Concluir Tarefa

URI: lgapontes.com/aulas/todoapp/api/tarefa/concluir

Method: PUT

```
Body: {  
  "codigo": 3  
}
```

PUT localhost/todoapp/api/tarefa/concluir

Send

200 OK

TIME 1.67 s

SIZE 46 B



JSON

Auth

Query

Header 1

Docs

```
1 {  
2   "codigo": 3  
3 }
```

Preview

Header 4

Cookie

Timeline

```
1 {  
2   "codigo": "3",  
3   "nome": "Luke",  
4   "experiencia": "2"  
5 }
```

TO-DO App

API para Apagar Tarefa

URI: lgapontes.com/aulas/todoapp/api/tarefa/apagar

Method: DELETE

```
Body: {  
  "codigo": 3  
}
```

DELETE ▼ lgapontes.com/aulas/todoapp/api/tarefa/apagar

Send

200 OK

TIME 272 ms

SIZE 46 B



JSON ▼

Auth ▼

Query

Header 1

Docs

```
1 ▼ {  
2   "codigo": 5  
3 }
```

Preview ▼

Header 5

Cookie

Timeline

```
1 ▼ {  
2   "codigo": "3",  
3   "nome": "Luke",  
4   "experiencia": "0"  
5 }
```

Debate em sala

- **Registrar timestamp da nova tarefa:** no front-end (JavaScript) ou no back-end (Servidor PHP)?
- **Regras de Negócio:** no front-end (JavaScript) ou no back-end (Servidor PHP)?

Headers Cookies Params Response Timings

Request URL: `http://lgapontes.com/aulas/todoapp/api/tarefas`

Request method: GET

Remote address: 192.185.213.160:80

Status code: 200 OK ? **Edit and Resend** Raw headers

Version: HTTP/1.1

Filter headers

Response headers (253 B)

Request headers (359 B)



O Firefox já possui por padrão este recurso. Os demais navegadores possuem plugins.

É possível manipular o *Request* pelos navegadores

New Request **Send** Cancel

POST `http://lgapontes.com/aulas/todoapp/api/tarefa`

Request Headers:

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Connection: keep-alive
Host: lgapontes.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0

Request Body:

```
{
  "codigoUsuario": 3,
  "codigoTipo": 2,
  "texto": "Correr na praia"
  "timestamp": "08/05/2018 00:00:00"
}
```

POST Nova Tarefa (exe13.html)

```
var tarefa = { codigoUsuario: 1, codigoTipo: 1, texto: 'Nova tarefa!' };
var json = JSON.stringify(tarefa);
var xhr = new XMLHttpRequest();
xhr.open('POST', 'http://lgapontes.com/aulas/todoapp/api/tarefa');
xhr.setRequestHeader("Content-Type", "application/json");
xhr.addEventListener('load', function(){
    var usuario = JSON.parse(xhr.responseText);
    console.log(usuario);
});
xhr.send(json);
```

Exercício (exe12/index.html)

Vamos alterar agora o comportamento da funcionalidade de login. Ao invés de simplesmente mostrar o formulário de cadastro de tarefas, ele deverá realizar um POST para a API de login passando pelo `send()` o JSON com o nome do usuário. Esta API retorna um JSON do usuário com todas as suas tarefas. A partir dessa lista, deve-se fazer um *forEach()* para inclui-las na área apropriada da página.

Dica: reuse a lógica de criar etapas, que é capaz de incluir os elementos necessários. Para testar, use como exemplo o usuário *Igapontes*.

Extra: Adicione a classe *is-loading* na `<div> control` do `<input>` do nome para aplicar um efeito de loading enquanto a API faz o login.

PUT Concluir Tarefa (exe14.html)

```
var tarefa = { codigo: 1 };  
var json = JSON.stringify(tarefa);  
var xhr = new XMLHttpRequest();  
xhr.open('PUT', 'http://localhost/todoapp/api/tarefa/concluir');  
xhr.setRequestHeader("Content-Type", "application/json");  
xhr.addEventListener('load', function(){  
    var usuario = JSON.parse(xhr.responseText);  
    console.log(usuario);  
});  
xhr.send(json);
```

DELETE Apagar Tarefa (exe15.html)

```
var tarefa = { codigo: 1 };  
var json = JSON.stringify(tarefa);  
var xhr = new XMLHttpRequest();  
xhr.open('DELETE', 'http://localhost/todoapp/api/tarefa/apagar');  
xhr.setRequestHeader("Content-Type", "application/json");  
xhr.addEventListener('load', function(){  
    var usuario = JSON.parse(xhr.responseText);  
    console.log(usuario);  
});  
xhr.send(json);
```

E se tentarmos apagar uma tarefa que não existe?

Código da tarefa que será apagada

999

Elements Console Sources Network Performance Memory >> 2

top Filter Default levels Group similar

- ✖ DELETE <http://localhost/todoapp/api/tarefa/apagar> 400 (Bad Request) <localhost/todoapp/api/tarefa/apagar:1>
- ✖ ▶ Uncaught SyntaxError: Unexpected end of JSON input
at JSON.parse (<anonymous>)
at XMLHttpRequest.<anonymous> (exe15.html:70) [VM24:1](#)

▼ General

Request URL: <http://localhost/todoapp/api/tarefa/apagar>

Request Method: DELETE

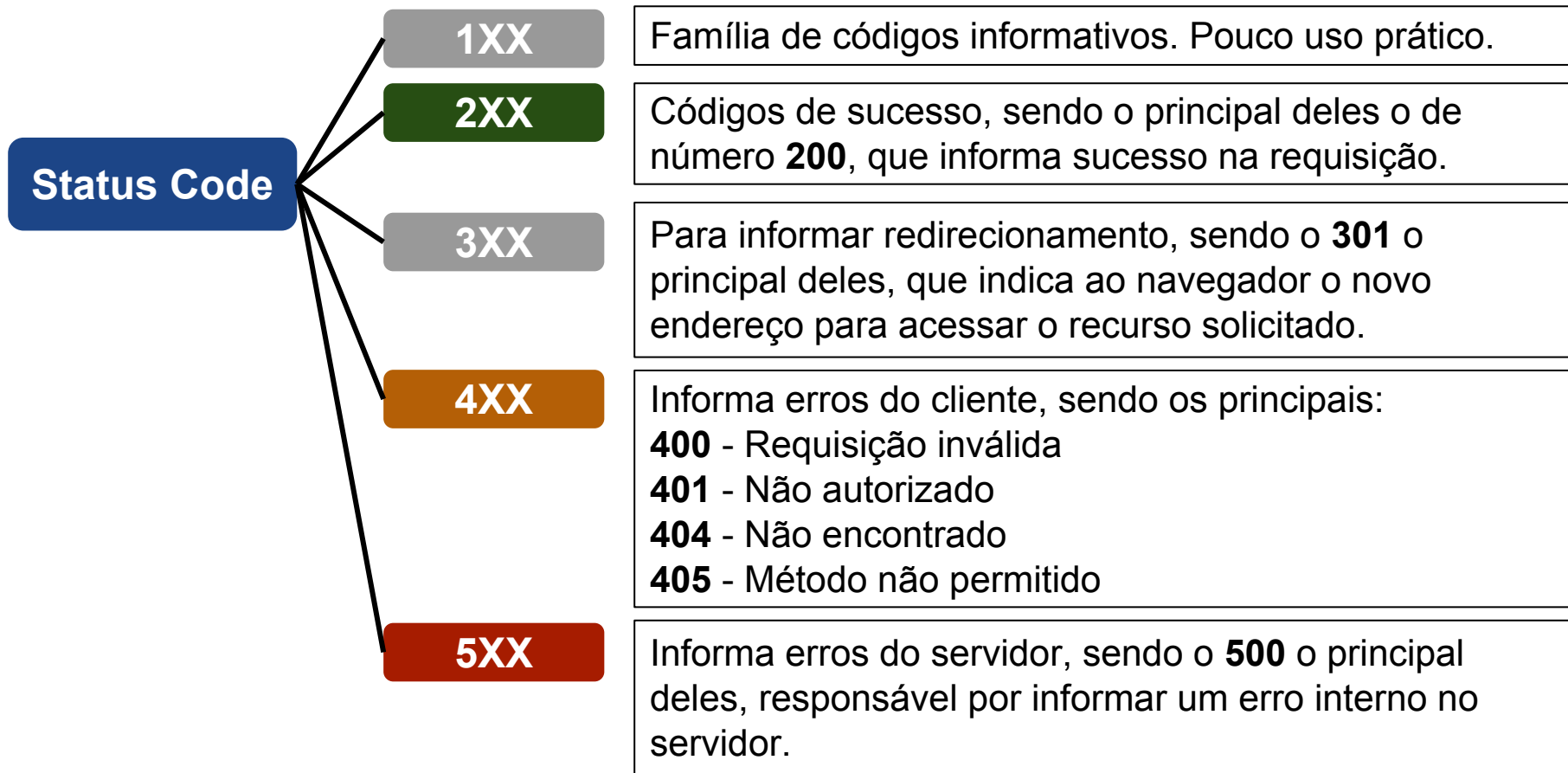
Status Code: ● 400 Bad Request

Remote Address: [::1]:80

Referrer Policy: no-referrer-when-downgrade

Códigos de status do HTTP

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes



Códigos de status do HTTP (exe16.html)



É uma boa prática utilizar os códigos de status HTTP para interagir com as APIs. Por exemplo, uma API de cadastro que só aceita o método **POST** deve retornar um erro 405 caso o usuário tente executar outro método (GET, PUT, etc).

```
xhr.addEventListener('load',function(){
  if (xhr.status == 200) {
    console.log( JSON.parse(xhr.responseText) );
  } else {
    console.log(xhr.status + ' : ' + xhr.statusText);
  }
});
```



Sempre verifique se o status é 200 antes de tratar o retorno da API.

Códigos de status do XMLHttpRequest (exe17.html)

Além do código de status do HTTP, podemos também verificar o **readyState** do objeto XMLHttpRequest. Isso vai garantir que o carregamento foi realizado antes de obter o resultado.

Vide: <https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest/readyState>

Valor	Estado	Descrição
0	UNSENT	Um cliente foi criado. Mas o método <code>open()</code> não foi chamado ainda.
1	OPENED	O método <code>open()</code> foi chamado.
2	HEADERS_RECEIVED	o método <code>send()</code> foi chamado e os cabeçalhos e status estão disponíveis .
3	LOADING	Baixando e <code>responseText</code> contem os dados parciais.
4	DONE	Operação concluída.

Códigos de status do XMLHttpRequest (exe17.html)

Na prática precisamos verificar se o **readyState** é igual a 4.

```
xhr.addEventListener('load',function(){  
    if (xhr.readyState == 4 && xhr.status == 200) {  
        console.log( JSON.parse(xhr.responseText) );  
    } else {  
        console.log(xhr.status + ': ' + xhr.statusText);  
    }  
});
```

Escutando erros e timeout (exe18.html)

Por fim, para tratarmos outros possíveis erros ou *timeout*, podemos escutar também outros dois eventos:

```
xhr.addEventListener('timeout',function(){  
    console.log('Não foi possível obter o conteúdo!');  
});  
xhr.addEventListener('error',function(evento){  
    console.log('Ocorreu um erro ao obter o conteúdo!');  
});
```

É possível definir um tempo (em milisegundos) de timeout com a sintaxe:

```
xhr.timeout = 3000;
```

Exercício (exe19/index.html)

- Utilize a API de nova tarefa para registrar as tarefas criadas via formulário de criação das tarefas.
- Utilize a API de concluir e apagar tarefas a partir dos respectivos botões disponíveis nas tarefas.
- A API do TO-DO App sempre retorna um JSON com os dados atuais do usuário, incluindo os pontos de experiência. Utilize essa informação para atualizar o total de experiência no cabeçalho da página.

Importante: faça a validação do status code 200.

Dica: guarde o código do usuário logado e das tarefas criadas em um `<input type="hidden">`

Controlando % de download (exe20.html)

Através do evento *progress* é possível obter 3 propriedades a partir das quais nós podemos controlar o progresso da operação.

lengthComputable	Valor booleano que indica se é possível ou não verificar o total de bytes transferidos.
loaded	Número de bytes que já foram transferidos.
total	Total de bytes que será transferido durante a operação.

Para testar, acesse: <http://lgapontes.com/aulas/js/progress/>

Obrigado!