

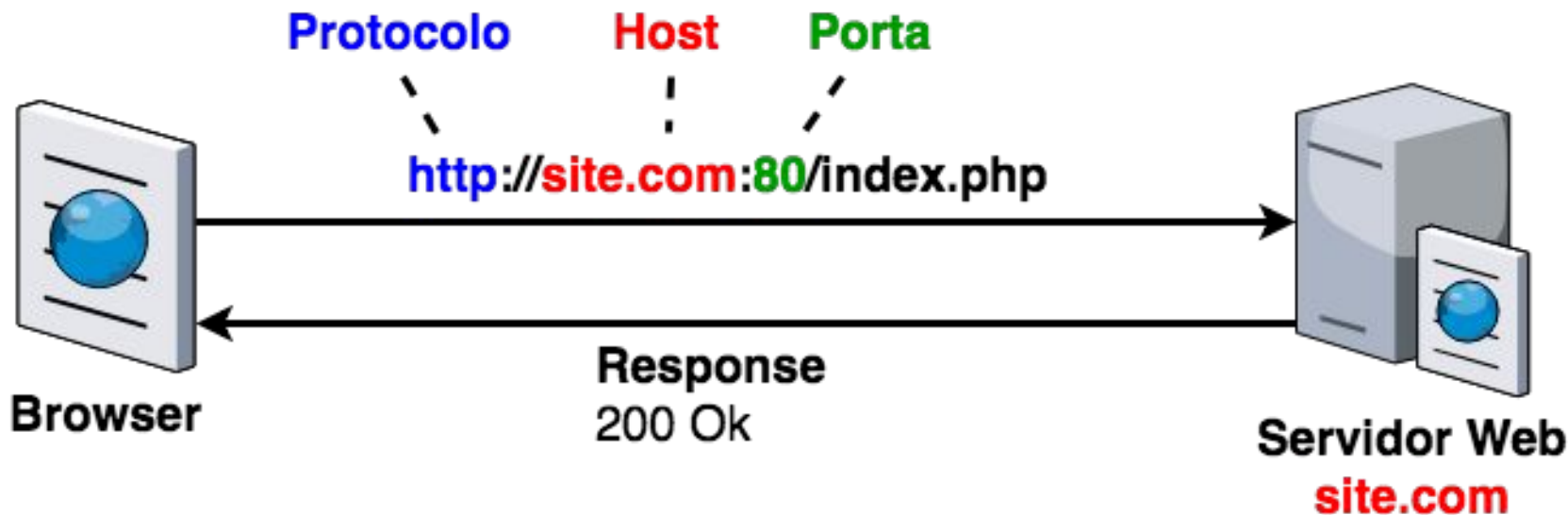
# PHP

## Aula 4

# Same Origin Policy

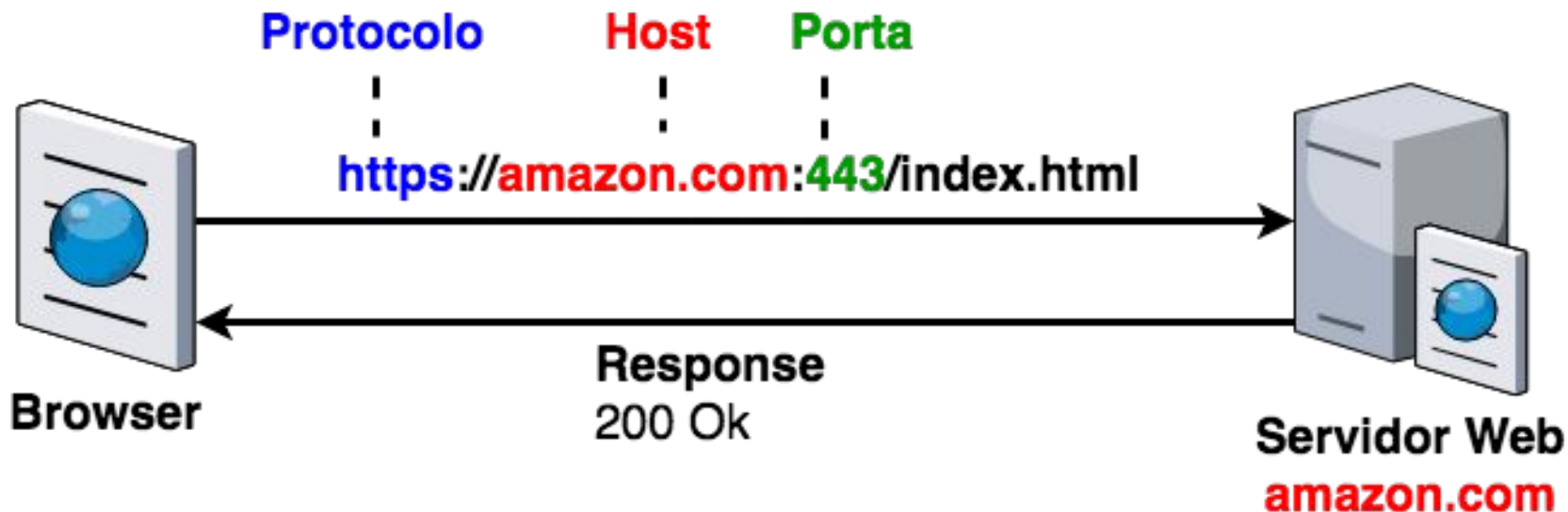
# Same Origin Policy

Em uma requisição tradicional, o browser acessa um recurso do servidor através do **protocolo**, do **host** e da **porta**. Com isso o servidor web responde ao browser enviando-lhe o recurso (index.php) acessado.



# Same Origin Policy - Exemplo real

Suponha por exemplo que um usuário esteja acessando o site <https://amazon.com>. Neste exemplo, o *Amazon* trabalha com o protocolo *https* e a porta *443*.

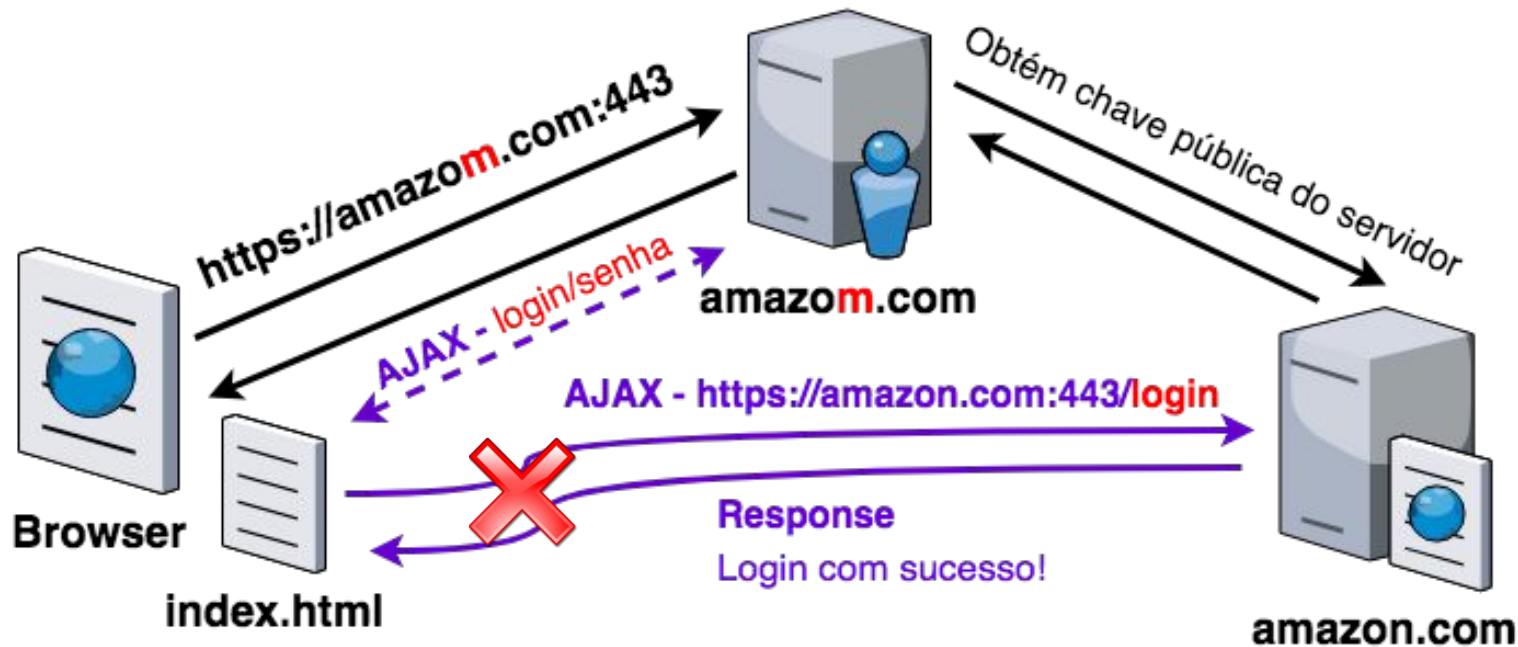


# Same Origin Policy - AJAX

Depois que a página já está no browser, é comum que ela realize várias solicitações assíncronas (AJAX) para obter outros dados do servidor web. Neste exemplo, veja que esta consulta AJAX é feita sob o mesmo protocolo, host e porta do recurso original.



# Same Origin Policy - *Man-in-the-Middle*

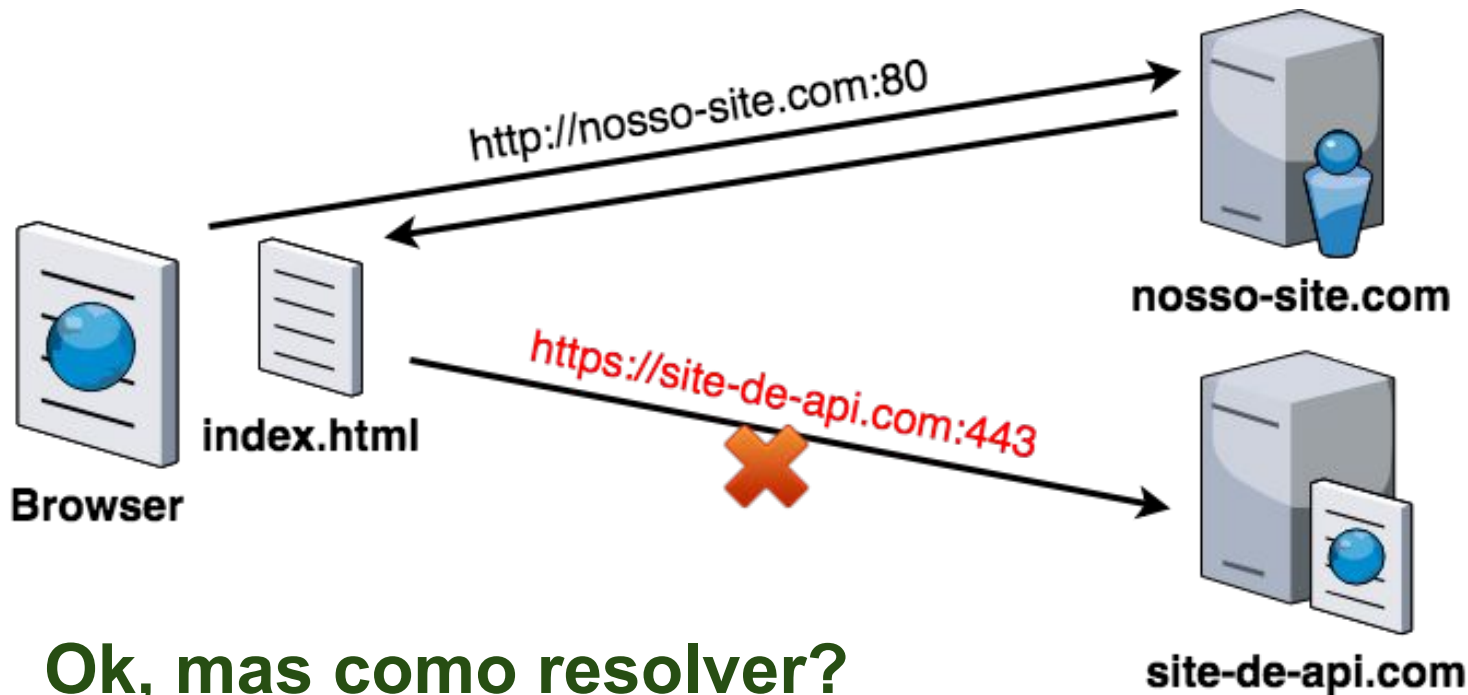


Neste caso, o browser possui uma página fornecida por **https://amazom.com:443** e posteriormente faz uma chamada AJAX ao site **https://amazon.com:443**. Esse tipo de ataque é **atenuado** porque os navegadores implementam um tipo de segurança chamado *Same Origin Policy*.

**Ok, mas por que isso é importante para nós?**

# Same Origin Policy

Porque geralmente APIs são providas para que qualquer site possa consumir. Neste caso, a **Same Origin Policy** também bloqueará o acesso.



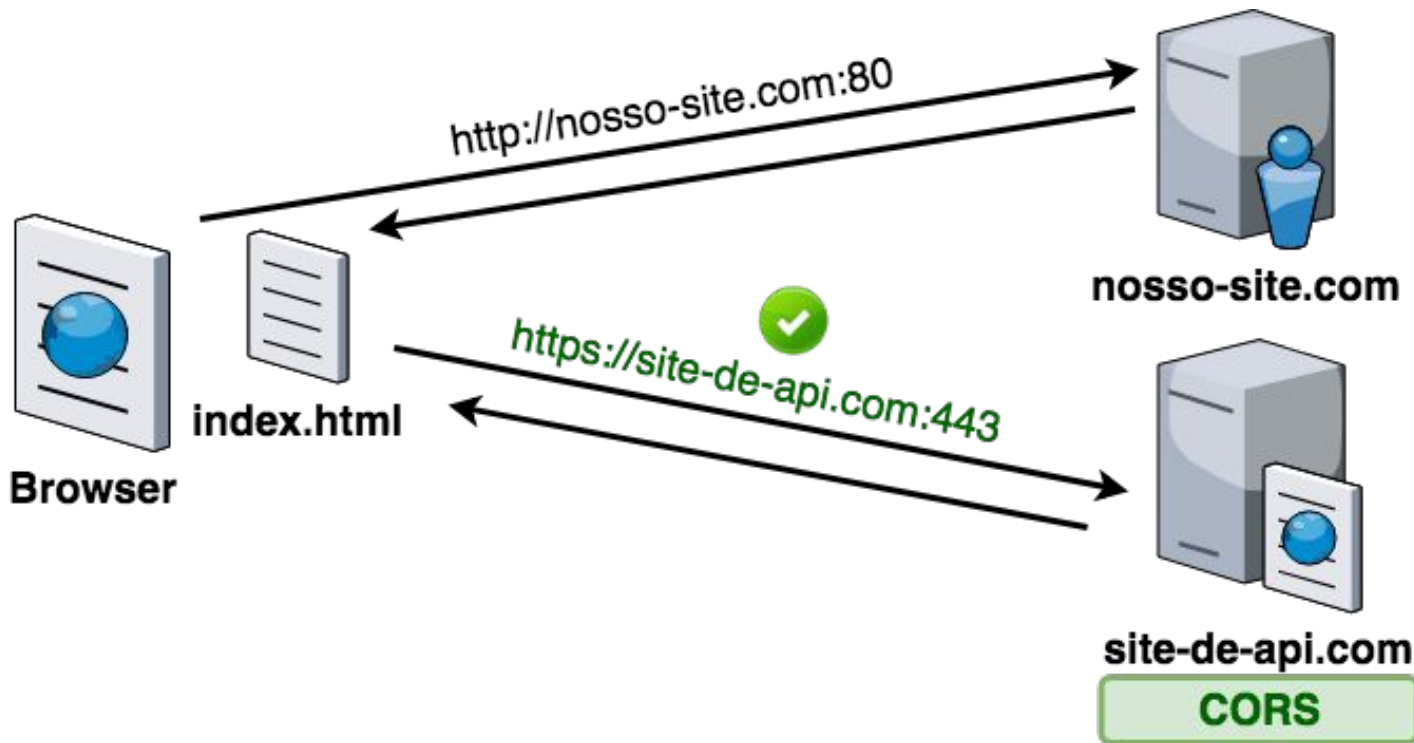
Ok, mas como resolver?



# Cross-Origin Resource Sharing (CORS)

# Cross-Origin Resource Sharing (CORS)

O CORS é um recurso configurado no **servidor** para avisar aos browsers que determinado site **aceita** requisições de outras origens.



# CORS no PHP (exe1/index.php)

Você pode habilitar os CORS no **PHP** ou no **Apache**. Em ambos os casos você pode habilitar para uma determinada origem (protocolo, host e porta) ou liberar o acesso para todas as origens.

No PHP, você deve acrescentar o cabeçalho *Access-Control-Allow-Origin* através da função *header()*.



Veja como liberar para uma única origem.

```
header('Access-Control-Allow-Origin: http://127.0.0.1:8080');
```

Veja como liberar para todas as origens.

```
header('Access-Control-Allow-Origin: *');
```

# CORS no Apache (exe1/index.php)

No caso do Apache, deve-se criar um arquivo **.htaccess** no diretório onde estão localizados os arquivos PHP da API. De forma semelhante ao PHP, devemos acrescentar o cabeçalho *Access-Control-Allow-Origin*, conforme abaixo.

Veja como liberar para uma única origem.

```
Header set Access-Control-Allow-Origin "http://127.0.0.1:8080"
```

Veja como liberar para todas as origens.

```
Header set Access-Control-Allow-Origin "*"
```



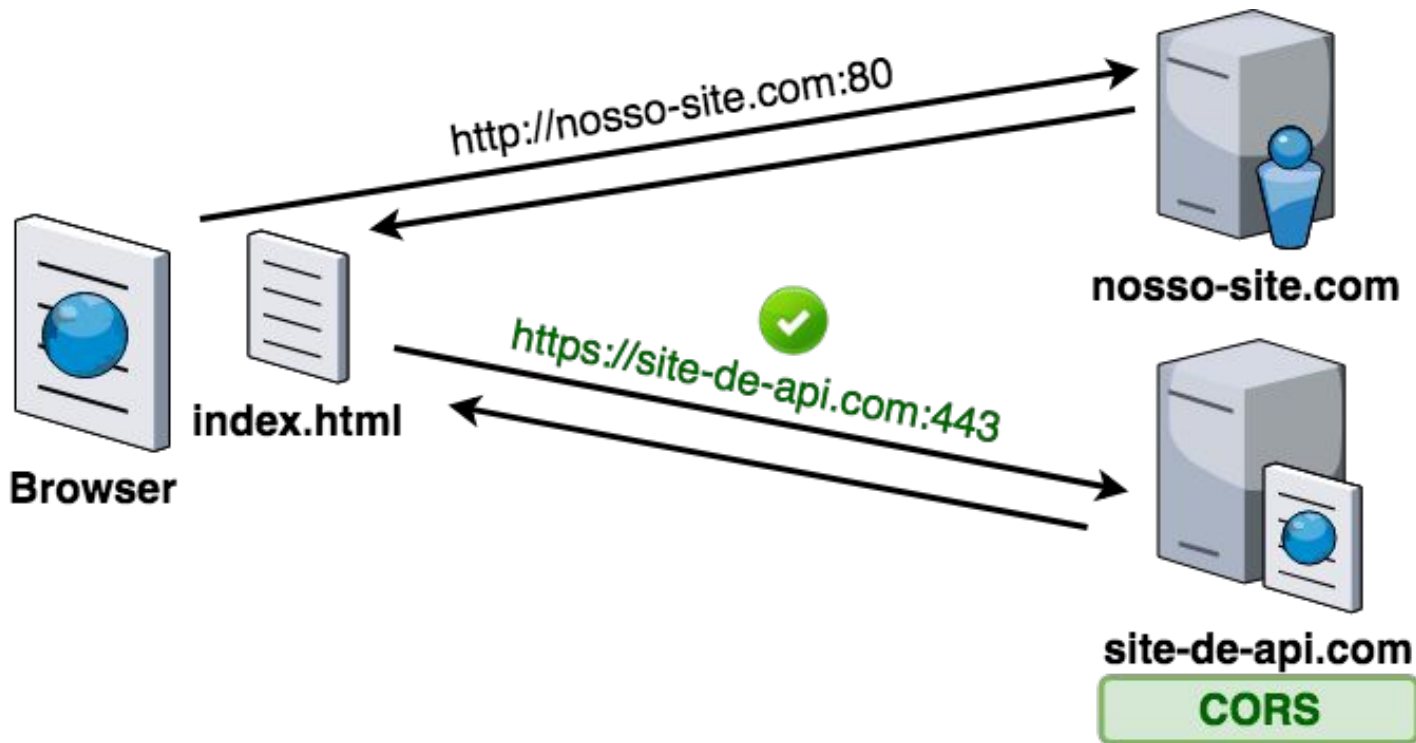
**Apache**



Em servidores reais de produção deve-se optar pela configuração no Apache (ao invés de no PHP), pois ele pode servir de "ponte" para outras linguagens.

# Cross-Origin Resource Sharing (CORS)

Então, sempre que você for publicar uma API, tenha em mente que caso você queira disponibilizá-la para outros sites, o CORS deve ser habilitado.



# Construindo uma API

# Construindo uma API (exe2.php)

Vamos começar construindo uma API que apenas retorna uma string. Veja a seguir um exemplo de código.

Exemplo bem simples!

```
<?php
    header('Access-Control-Allow-Origin: *');

    echo "Minha primeira API!";
    die();
```

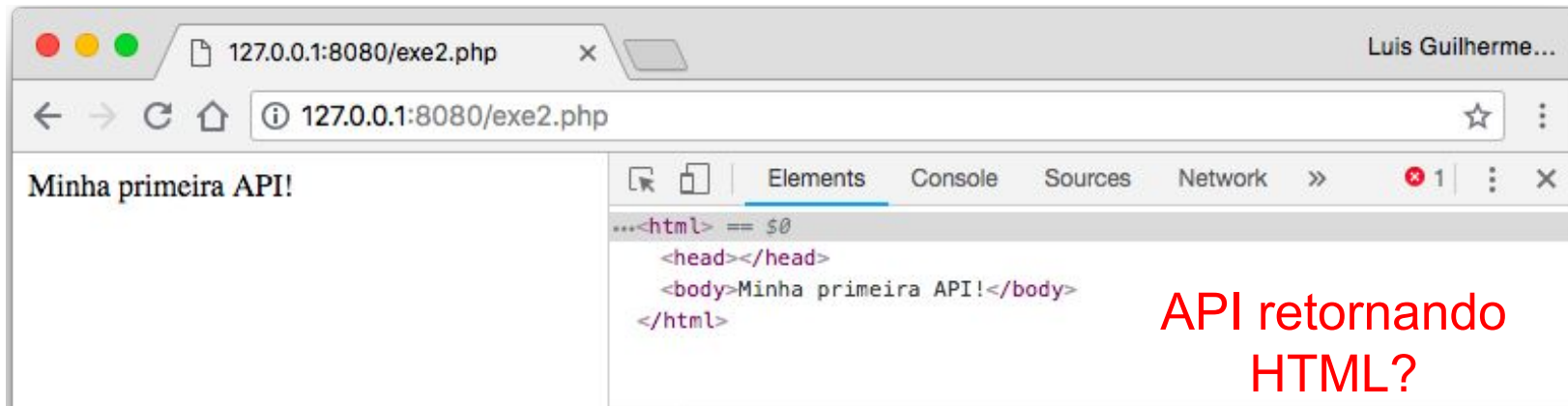


É uma boa prática invocar a função *die()* após o retorno da API. Isso garante que o script PHP seja interrompido neste ponto.



Este código não está implementado de forma correta. Mais detalhes no próximo slide.

# Construindo uma API (exe2.php)



Podemos verificar na aba *Elements* do *DevTools* que esta API está retornando uma página HTML.

**Ok, mas qual o problema disso?**

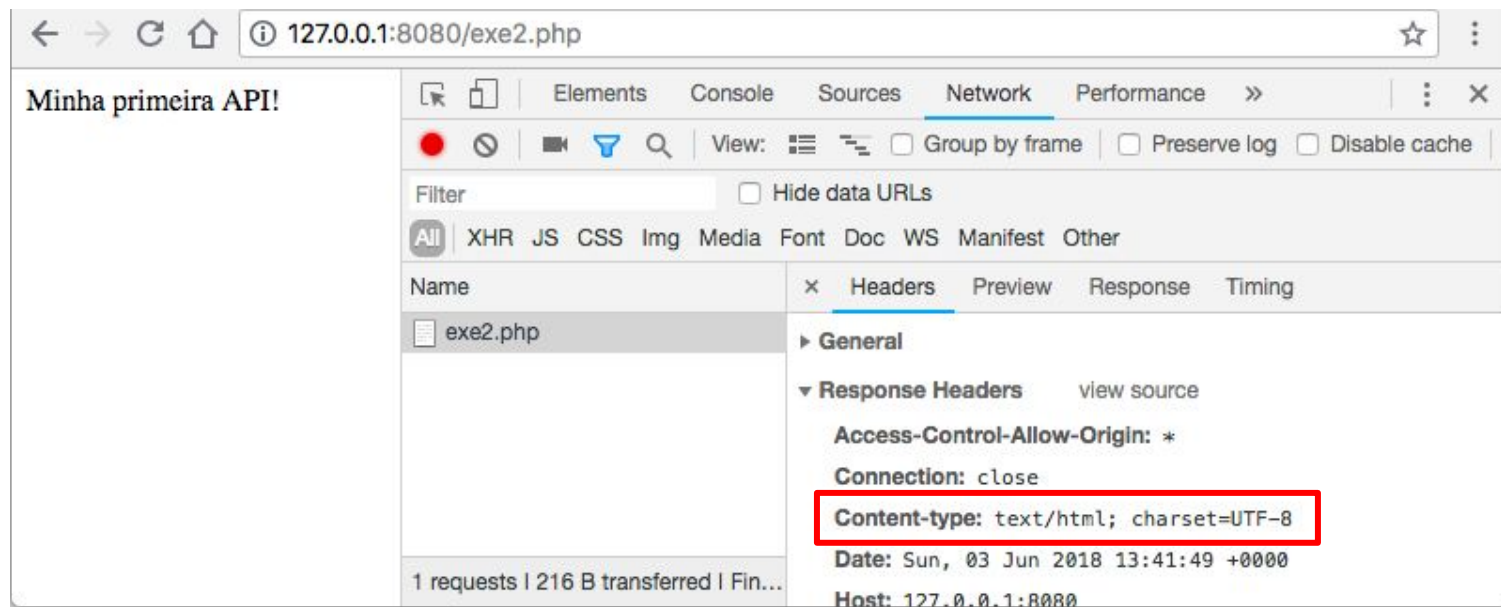
APIs são construídas para tráfego de dados. Não é correto retornar corpo HTML no **Response** de uma API.

**Então como podemos acertá-la?**



# Construindo uma API (exe2.php)

Na **Response** (resposta retornada) da API há um **header** chamado **Content-Type**, que por default vem preenchido com o *MIME type* **text/html**.



Sabendo disso, podemos então ajustar o script PHP para ao invés de retornar **text/html**, passar a retornar um **application/json**.

# Construindo uma API (exe3.php)

A configuração do cabeçalho *Content-Type* no PHP é muito simples. No nosso caso, vamos optar em trabalhar com o *MIME Type* **application/json**, que é um tipo adequado para tráfego de dados entre os browser e o servidor.

Exemplo bem simples com *Content-Type* definido como **application/json**

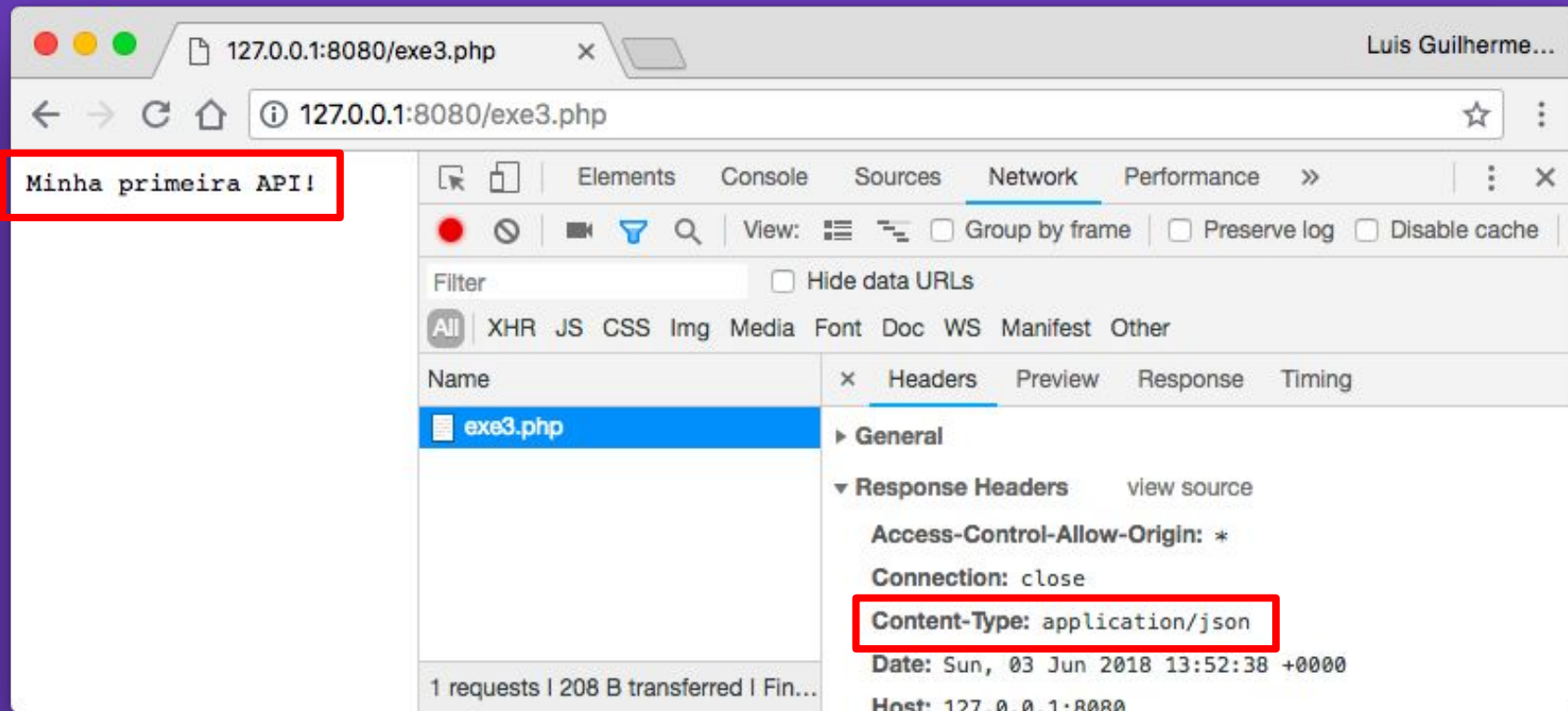
```
<?php
    header('Access-Control-Allow-Origin: *');

    header('Content-Type: application/json');
    echo "Minha primeira API!";
    die();
```



Ainda há um pequeno erro aqui!

# O que vocês acham que está errado neste retorno?



# Estrutura de um JSON

O problema é que atualmente nossa API ainda não está retornando um JSON estruturado corretamente. Um JSON é uma estrutura de dados baseada em chaves e valores. Veja abaixo um exemplo retirado do W3C.

## JSON

```
{ "name": "John" }
```

Para mais detalhes: [https://www.w3schools.com/js/js\\_json\\_syntax.asp](https://www.w3schools.com/js/js_json_syntax.asp)

# Construindo uma API (exe4.php)

No PHP é muito simples converter *arrays* ou *objetos* em uma estrutura JSON válida. Para isso, precisamos apenas utilizar a função `json_encode()`

Convertendo um **array** para JSON

```
<?php
    header('Access-Control-Allow-Origin: *');

    $dados = array(
        "mensagem" => "Minha primeira API!"
    );

    header('Content-Type: application/json');
    echo json_encode($dados);
    die();
```

# Construindo uma API (exe5.php)

Convertendo um *objeto* para JSON

```
<?php
    header('Access-Control-Allow-Origin: *');

    class Dados {
        public $mensagem;
        public function __construct($mensagem) {
            $this->mensagem = $mensagem;
        }
    }

    $dados = new Dados("Minha primeira API!");

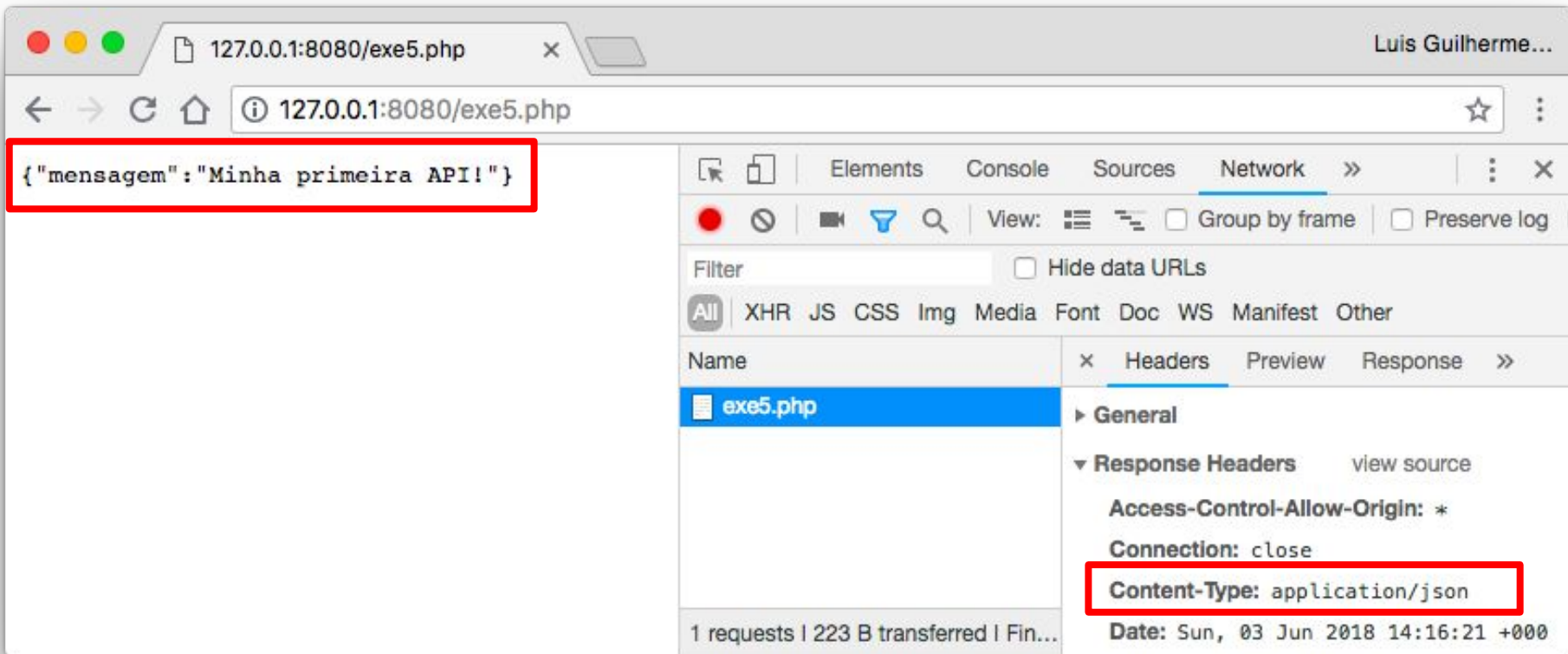
    header('Content-Type: application/json');
    echo json_encode($dados);
    die();
```



Veremos mais detalhes da conversão de objetos em breve.

# Construindo uma API (exe4.php e exe5.php)

Em ambos os casos o retorno é:



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8080/exe5.php`. The page content shows a JSON response: `{"mensagem": "Minha primeira API!"}`, which is highlighted with a red rectangle. The browser's developer tools are open to the Network tab, showing a single request for `exe5.php`. The request details are expanded, showing the 'Response Headers' section. The 'Content-Type' header is highlighted with a red rectangle and set to `application/json`. Other visible headers include 'Access-Control-Allow-Origin: \*' and 'Connection: close'. The status bar at the bottom indicates '1 requests | 223 B transferred | Fin...'. The browser's title bar shows 'Luis Guilherme...'.

```
{ "mensagem": "Minha primeira API!" }
```

Network tab details for `exe5.php`:

- General**
- Response Headers**
  - Access-Control-Allow-Origin: \*
  - Connection: close
  - Content-Type: application/json**
  - Date: Sun, 03 Jun 2018 14:16:21 +000

1 requests | 223 B transferred | Fin...

# Retorno no formato unicode (exe6.php)

Para evitar erros de acentuação na conversão do JSON, devemos informar ao PHP o uso da constante JSON\_UNESCAPED\_UNICODE

Conversão com JSON\_UNESCAPED\_UNICODE

```
$dados = new Dados("API sem erros de acentuação!");  
header('Content-Type: application/json');  
echo json_encode($dados, JSON_UNESCAPED_UNICODE);  
die();
```

Ao utilizar a conversão com unicode, teremos o retorno correto dos dados.



```
{"mensagem": "API sem erros de acentua\u00e7\u00e3o!"}
```



```
{"mensagem": "API sem erros de acentuação!"}
```



# Exercício (exe7.php)

Crie uma simples API estática (sem acesso a banco de dados) que retorne os dados de um cliente no seguinte formato:

```
{  
    "nome": "João",  
    "idade": 40,  
    "aposentado": false  
}
```

**Observação:** pode usar a notação de *arrays* ou *classes*.

# Exercício (exe7.php)

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8080/exe7-com-arrays.php`. The page content displays a JSON object: `{"nome": "João", "idade": 40, "aposentado": false}`.

The browser's developer tools are open, showing the **Network** tab. A request to `exe7-com-arrays.php` is selected. The **Headers** sub-tab is active, displaying the following response headers:

- Access-Control-Allow-Origin:** \*
- Connection:** close
- Content-Type:** application/json
- Date:** Sun, 03 Jun 2018 14:42:38 +000

The status bar at the bottom indicates: 1 requests | 235 B transferred | Fin...

# APIs de consulta de dados

## Listar perfis (exe8/index.php)

Uma API de verdade geralmente trabalha com banco de dados. Após realizar a consulta, ao invés de manipular HTML, ela deve retornar um JSON.

**PASSO 1:** implemente um repositório para realizar as consultas no banco.

Repositório para retornar uma lista de *perfis*

```
$perfis = array();
$conexao = conectar();
$query = $conexao->prepare('select codigo, descricao from perfis');
$query->execute();
$resultado = $query->get_result();
if ($resultado->num_rows > 0) {
    while($perfil = $resultado->fetch_assoc()) {
        array_push($perfis,$perfil);
    }
}
return $perfis;
```



Note que nós podemos retornar um *array()* com os dados do banco diretamente.

# Listar perfis (exe8/index.php)

**PASSO 2:** o *array()* retornado pelo repositório pode ser diretamente convertido em JSON através da função *json\_encode()*

API para retornar os dados

```
<?php
    require_once "repositorio.php";

    header('Access-Control-Allow-Origin: *');

    $perfis = listarPerfis();

    header('Content-Type: application/json');
    echo json_encode($perfis, JSON_UNESCAPED_UNICODE);
    die();
```



**NUNCA** retorne dados sensíveis (como uma senha, por exemplo) através de uma API.

## Consultando um perfil (exe9/index.php)

Podemos também construir uma API para retornar um registro específico. Neste caso vamos utilizar *query strings* na URL para informar o código.

**PASSO 1:** crie um método no repositório para realizar consultas pelo código.

Repositório para retornar um perfil (ou nulo, se não encontrar)

```
$conexao = conectar();  
$query = $conexao->prepare('select codigo, descricao from perfis where codigo = ?');  
$query->bind_param('i',$codigo);  
$query->execute();  
$resultado = $query->get_result();  
if ($resultado->num_rows == 1) {  
    return $resultado->fetch_assoc();  
} else {  
    return null;  
}
```

## Consultando um perfil (exe9/index.php)

**PASSO 2:** em seguida, crie um script que a partir de uma verificação da *query string* **codigo** faz a consulta adequada no repositório.

API para retornar um ou todos os perfis de acordo com os dados recebidos por *query string*. Em outras palavras, se existir `$_GET["codigo"]`, o script buscará apenas um perfil. Caso contrário, retornará todos os perfis.

```
require_once "repositorio.php";

header('Access-Control-Allow-Origin: *');

if (isset($_GET['codigo'])) {
    $codigo = intval($_GET['codigo']);
    $json = obterPerfil($codigo);
} else {
    $json = listarPerfis();
}

header('Content-Type: application/json');
echo json_encode($json, JSON_UNESCAPED_UNICODE);
die();
```

# Consultando um perfil (exe9/index.php)

Da forma como está implementado, se o consumidor da API informar um código inválido ou inexistente, o script retorna apenas uma string `null`



É uma má prática **não** retornar informações para o consumidor da API.

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8080/exe9/index.php?codigo=50`. The page content displays `null`. The Chrome DevTools Network tab is open, showing a single request to `index.php?codigo=50`. The request details are visible, including the response headers:

- Access-Control-Allow-Origin:** \*
- Connection:** close
- Content-Type:** application/json
- Date:** Sun, 03 Jun 2018 16:31:39 +0000

The status bar at the bottom indicates: 1 requests | 193 B transferred | Finish: 7 ms | D...



# Retorno correto de uma API (exe10/index.php)

Há três formas de se implementar um retorno correto. Todas as três são aceitas pela comunidade, porém a mais sofisticada é a **terceira opção** (404 com texto).

## Recurso não encontrado

**Opção 1:** Envolver o retorno da API em um objeto (ou array) com informações gerais sobre a consulta. Esta é a técnica mais utilizada.

**Opção 2:** Simplesmente retornar um *Status Code 404* para o consumidor. O código 404 já informa que o recurso consultado não existe.

Veja: [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

**Opção 3:** Retornar um *Status Code 404* em conjunto com um JSON contendo a mensagem de erro. *Particularmente eu prefiro esta técnica.*

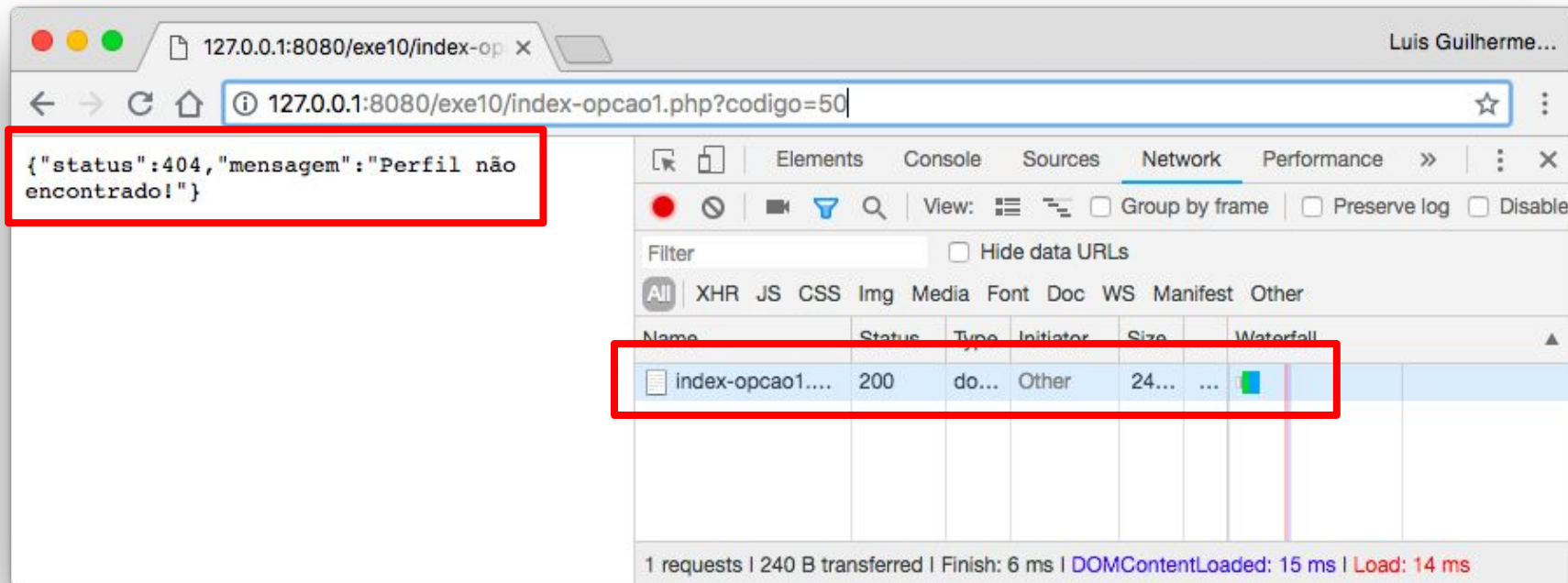
**Opção 1:** Envolver o retorno da API em um objeto (ou array) com informações gerais sobre a consulta. Esta é a técnica mais utilizada.

Trecho da opção 1 (veja *exe10/index-opcao1.php*)

```
$json = array();
$codigo = intval($_GET['codigo']);
$perfil = obterPerfil($codigo);
if ($perfil == null) {
    $json["status"] = 404;
    $json["mensagem"] = "Perfil não encontrado!";
} else {
    $json["status"] = 200;
    $json["mensagem"] = "Perfil retornado com sucesso!";
    $json["dados"] = $perfil;
}
header('Content-Type: application/json');
echo json_encode($json, JSON_UNESCAPED_UNICODE);
die();
```

# Retorno correto de uma API (exe10/index-opcao1.php)

Veja um retorno da **opção 1**:



**Opção 2:** Simplesmente retornar um *Status Code 404* para o consumidor. O código 404 já informa que o recurso consultado não existe.

Veja: [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

Trecho da opção 2 (veja *exe10/index-opcao2.php*)

```
$codigo = intval($_GET['codigo']);  
$json = obterPerfil($codigo);  
  
if ($json == null) {  
    header("HTTP/1.1 404");  
    die();  
}  
  
header('Content-Type: application/json');  
echo json_encode($json, JSON_UNESCAPED_UNICODE);  
die();
```

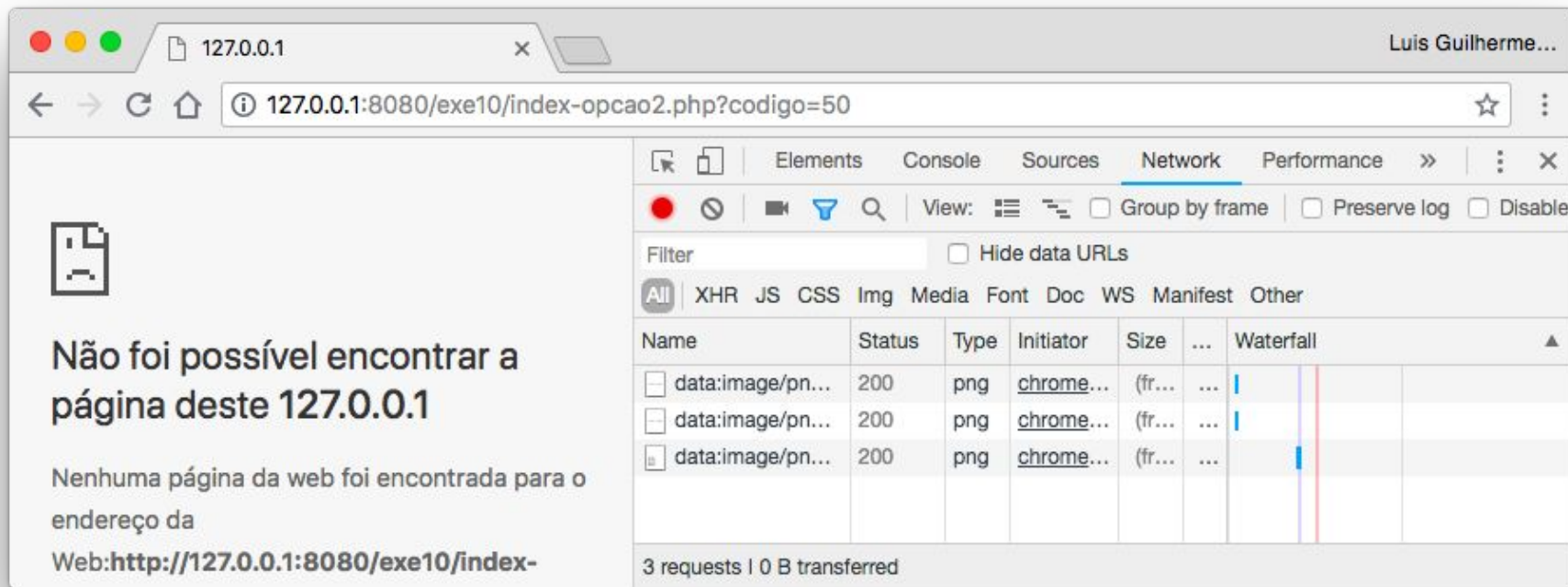
No PHP, para retornar um *Status Code* diferente de 200 (que é o default), devemos utilizar a função *header()* informando a versão do protocolo (neste caso, HTTP/1.1) seguida do código de status.

```
header("HTTP/1.1 404")
```

Sempre use ***die()*** após o retorno.

# Retorno correto de uma API (exe10/index-opcao2.php)

Veja um retorno da **opção 2**:



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8080/exe10/index-opcao2.php?codigo=50`. The main content area displays a 404 error message: "Não foi possível encontrar a página deste 127.0.0.1". Below the message, it states: "Nenhuma página da web foi encontrada para o endereço da Web: http://127.0.0.1:8080/exe10/index-".

The Chrome DevTools Network tab is open, showing a list of requests. The table below represents the data visible in the Network tab:

Name	Status	Type	Initiator	Size	...	Waterfall
data:image/pn...	200	png	chrome...	(fr...	...	
data:image/pn...	200	png	chrome...	(fr...	...	
data:image/pn...	200	png	chrome...	(fr...	...	

At the bottom of the Network tab, it indicates "3 requests | 0 B transferred".



Simplesmente retornar um 404 pode confundir o consumidor, fazendo-o pensar que a URL da API não existe.

**Opção 3:** Retornar um *Status Code 404* em conjunto com um JSON contendo a mensagem de erro. *Particularmente eu prefiro esta técnica.*

Trecho da opção 3 (veja *exe10/index-opcao3.php*)

```
$codigo = intval($_GET['codigo']);  
$json = obterPerfil($codigo);  
  
if ($json == null) {  
    header("HTTP/1.1 404");  
    echo '{"mensagem": "Perfil não encontrado!"}';  
    die();  
}  
  
header('Content-Type: application/json');  
echo json_encode($json, JSON_UNESCAPED_UNICODE);  
die();
```

Neste caso também devemos usar a função *header()*.

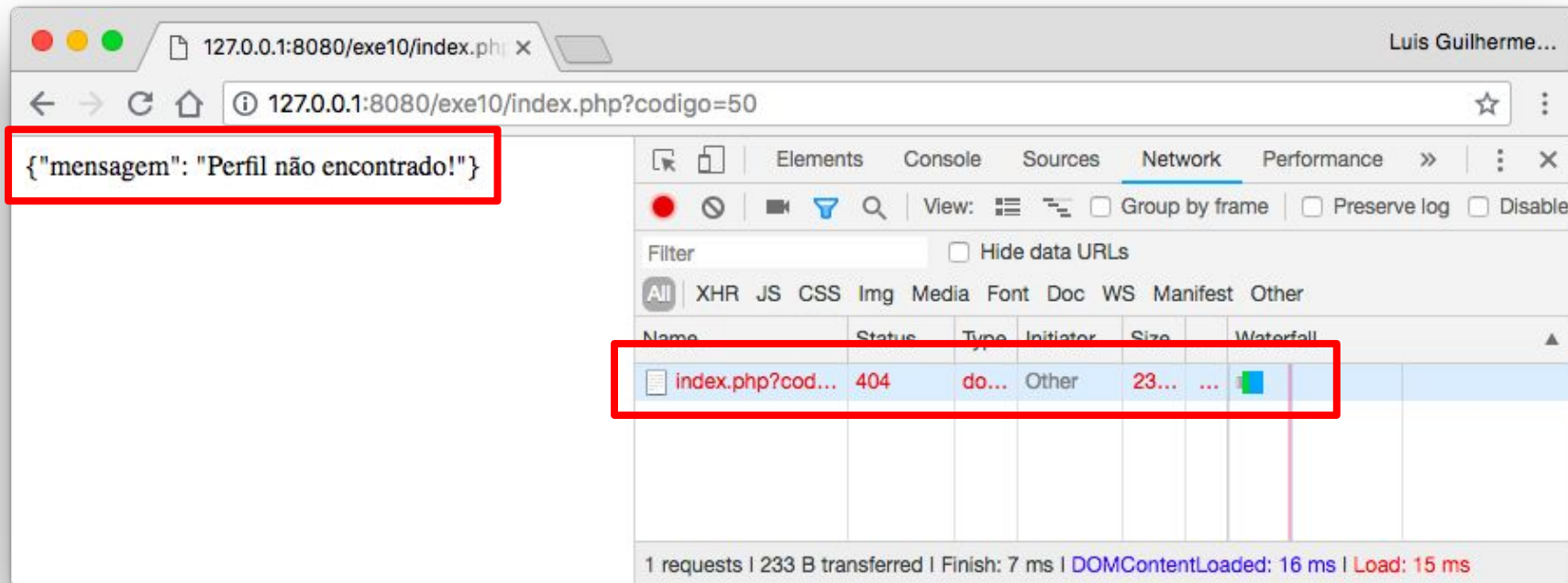
```
header("HTTP/1.1 404")
```

A diferença é que acrescentamos um corpo de retorno informando o motivo do 404.

O uso do ***die()*** também é recomendado.

# Retorno correto de uma API (exe10/index-opcao3.php)

Veja um retorno da **opção 3**:





# Consumindo a API (exe10/index.php)

O código *index.php* possui um exemplo de consumo da **opção 3**.

Veja um exemplo de código jQuery que está consultando a API (opção 3) de consulta de perfil. Neste caso, como a API está retornando um 404 quando o perfil não for encontrado, é necessário capturar o retorno no método *fail()*.

```
var url = 'http://127.0.0.1:8080/exe10/index-opcao3.php?codigo=' + $('input').val();
$.get(url, function(json){
    console.log(JSON.stringify(json));
}).fail(function(json){
    console.log(json.responseText);
});
```



# Exercício (exe11/index.php)

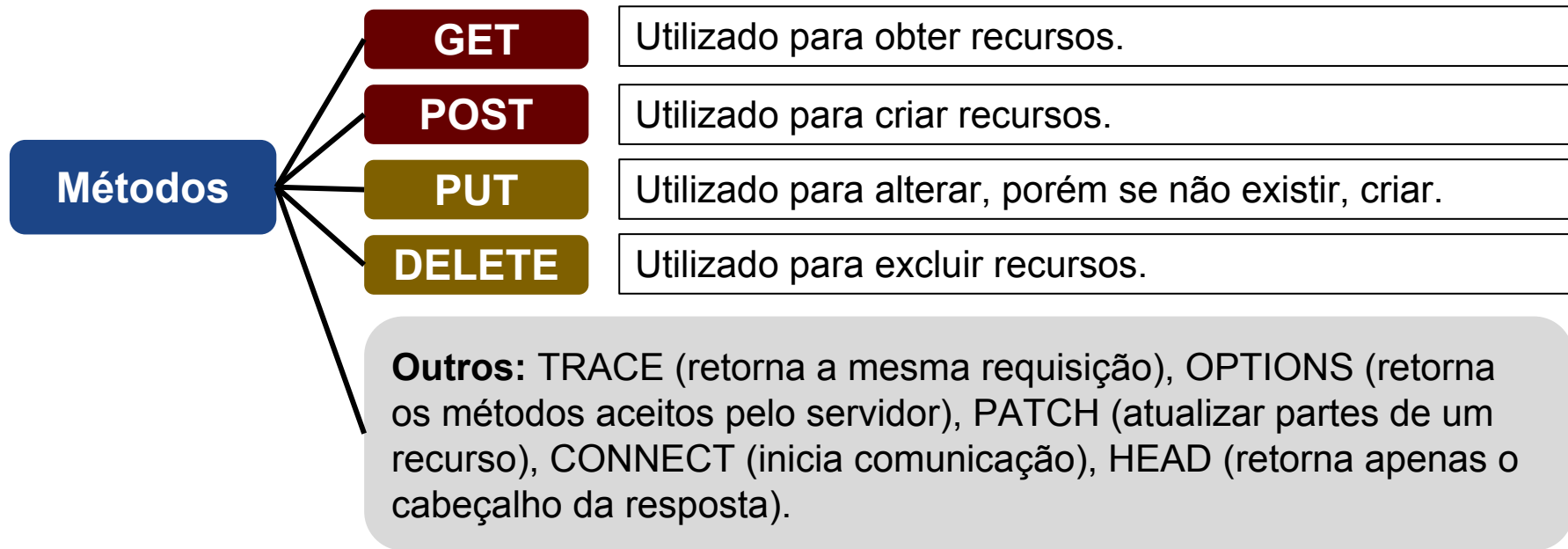
Construa um consumidor da **opção 1** da API apresentada nos slides anteriores. O código HTML e CSS pode ser idêntico ao do exercício 10.

Ajuste o código JavaScript para obter corretamente as respostas (de sucesso e falha) da API.

# Métodos HTTP e Códigos de Status

# Outros opções da API (métodos HTTP)

Até o momento nós apenas criamos uma API de consulta. É possível criar APIs para inclusão, alteração e exclusão de dados no banco de dados. A grande diferença é que devemos utilizar outros métodos HTTP.



## Outros opções da API (métodos HTTP)

### Idempotência

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

**Quais?**

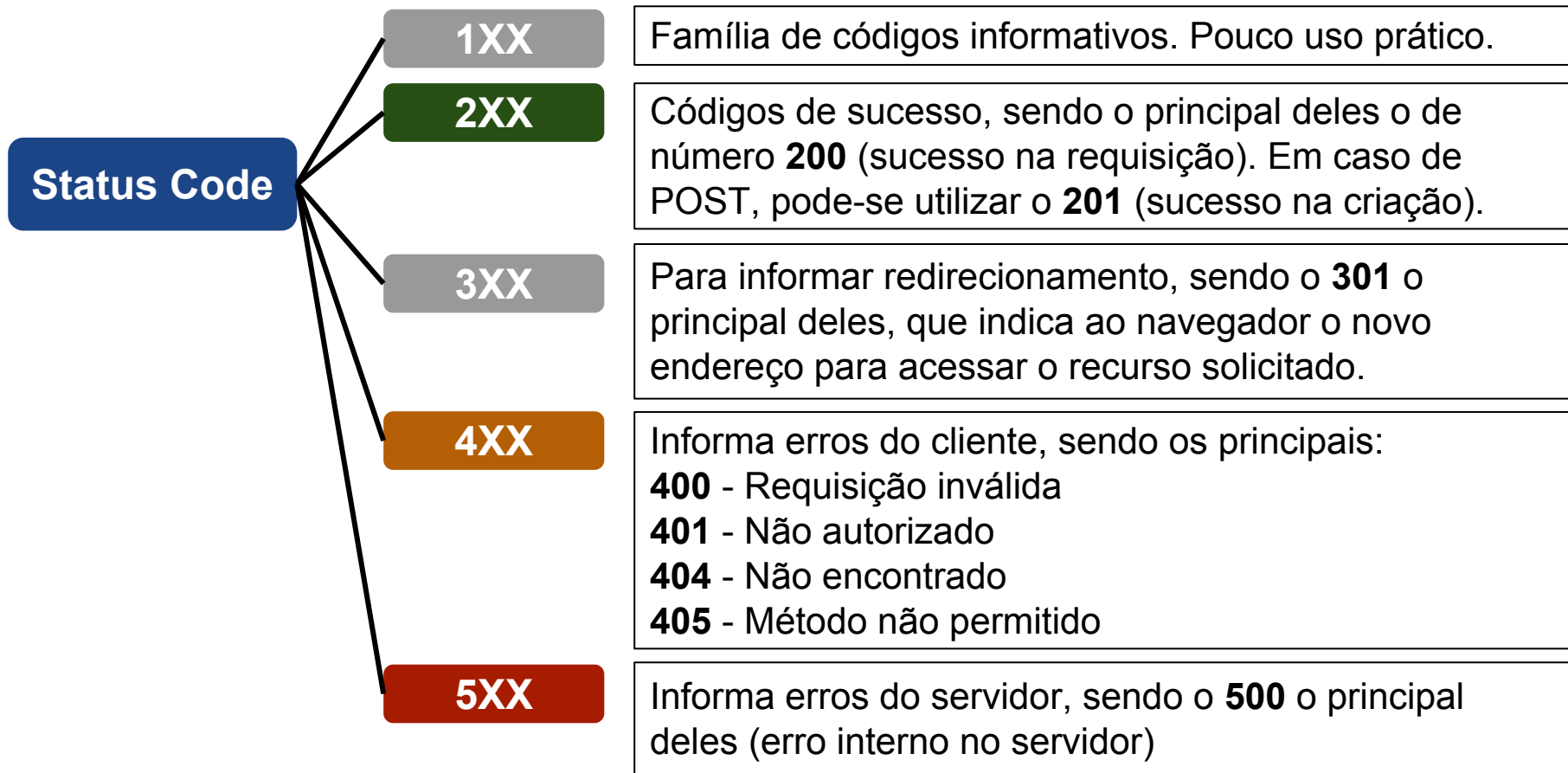
GET, OPTIONS, HEAD, PUT, TRACE, CONNECT e DELETE

**Exemplo:**

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

# Códigos de status do HTTP

[https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)



# Inserindo dados pela API (exe12/perfil.php)

Para implementar os outros métodos, precisamos tomar cuidado com alguns detalhes que podem tornar a API insegura (ou mal implementada). Vejamos:

## Detalhes iniciais:

Sempre use o método **POST** para inserir valores. Neste caso, é uma boa prática retornar o *Status Code* **201** caso o registro tenha sido inserido com sucesso.

## PASSO 1:

Verificar no PHP se o método utilizado pelo consumidor é o **POST**. Caso o método não seja este, retornar um *Status Code* **405**.

```
if ($_SERVER["REQUEST_METHOD"] === "POST") {  
    // Código ...  
} else {  
    header("HTTP/1.1 405");  
    echo '{"mensagem": "Método não permitido!"}';  
    die();  
}
```

# Inserindo dados pela API (exe12/perfil.php)

## PASSO 3:

Os dados serão enviados no **body** da solicitação. No PHP, para obter os dados do **body**, devemos utilizar a função `file_get_contents()` apontando para o local `php://input`. Este é o caminho a partir do qual o PHP obtém o **body**.

```
$body = file_get_contents("php://input");
```

## PASSO 4:

De posse do **body**, devemos convertê-lo em um `array()` do PHP através da função `json_decode()`.

```
$dados = json_decode($body, true);
```

A partir daí podemos utilizar `$dados["descricao"]` para obter a descrição do perfil.

## Inserindo dados pela API (exe12/perfil.php)

### PASSO 5:

É uma boa prática verificar a consistência dos dados enviados. Podemos fazer isso verificando com a função `array_key_exists()` se uma chave existe no `array()`. Se ela não existir, podemos retornar um *Status Code* **400**.

```
if ( array_key_exists("descricao",$dados) ) {  
    // Código...  
} else {  
    header("HTTP/1.1 400");  
    echo '{"mensagem": "Perfil não enviado corretamente!"}';  
    die();  
}
```



# Inserindo dados pela API (exe12/perfil.php)

## PASSO 6:

Por fim, com a descrição do perfil recuperada por `$dados["descricao"]`, podemos inserir o registro através do repositório. Neste caso, é importante encapsular a interação com o banco de dados em um *try/catch* pois caso ocorra alguma falha, devemos retornar um *Status Code 400* para o consumidor.

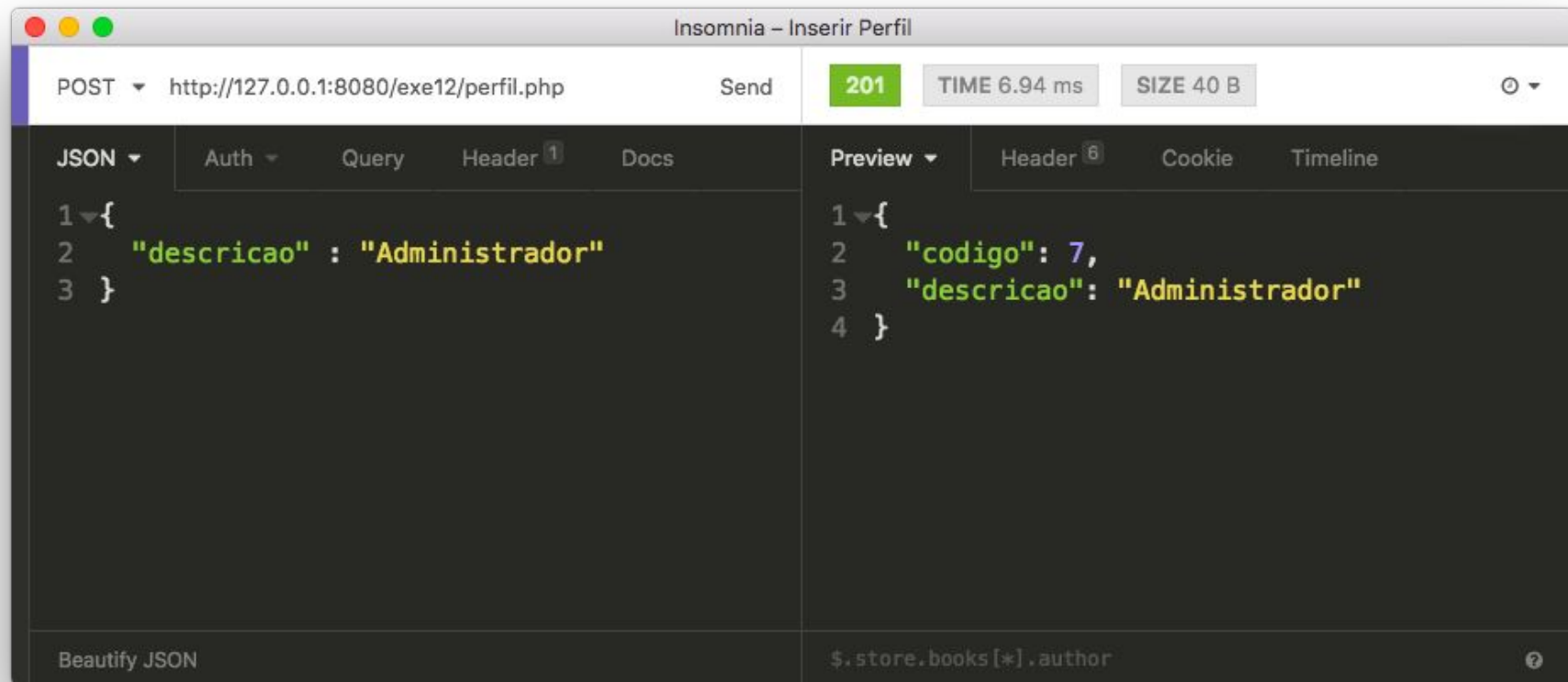
```
try {  
    $perfil = inserirPerfil($dados["descricao"]);  
    header('Content-Type: application/json');  
    header("HTTP/1.1 201");  
    echo json_encode($perfil, JSON_UNESCAPED_UNICODE);  
    die();  
} catch(Exception $e) {  
    header("HTTP/1.1 400");  
    echo '{"mensagem": "' . $e->getMessage() . '"}';  
    die();  
}
```

## Boa prática:

Para evitar que o consumidor faça uma nova consulta à API com o objetivo de obter o dado que acabou de ser inserido, é uma boa prática retornar o registro inserido como resposta no próprio POST.

# Inserindo dados pela API (exe12/perfil.php)

Vejamos um exemplo de inserção através do aplicativo **Insomnia REST**.



# Discussão em Sala

- Vocês acham válido realizar validações de regra de negócio no JSON recebido pelo POST antes de inseri-lo no banco de dados?
- Neste caso, se uma regra de negócio não for atendida, qual o *Status Code* mais apropriado para retorno?

Dica: [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

# Atualizando dados pela API (exe13/perfil.php)

A atualização de dados via API é semelhante à inserção. Veja as diferenças:

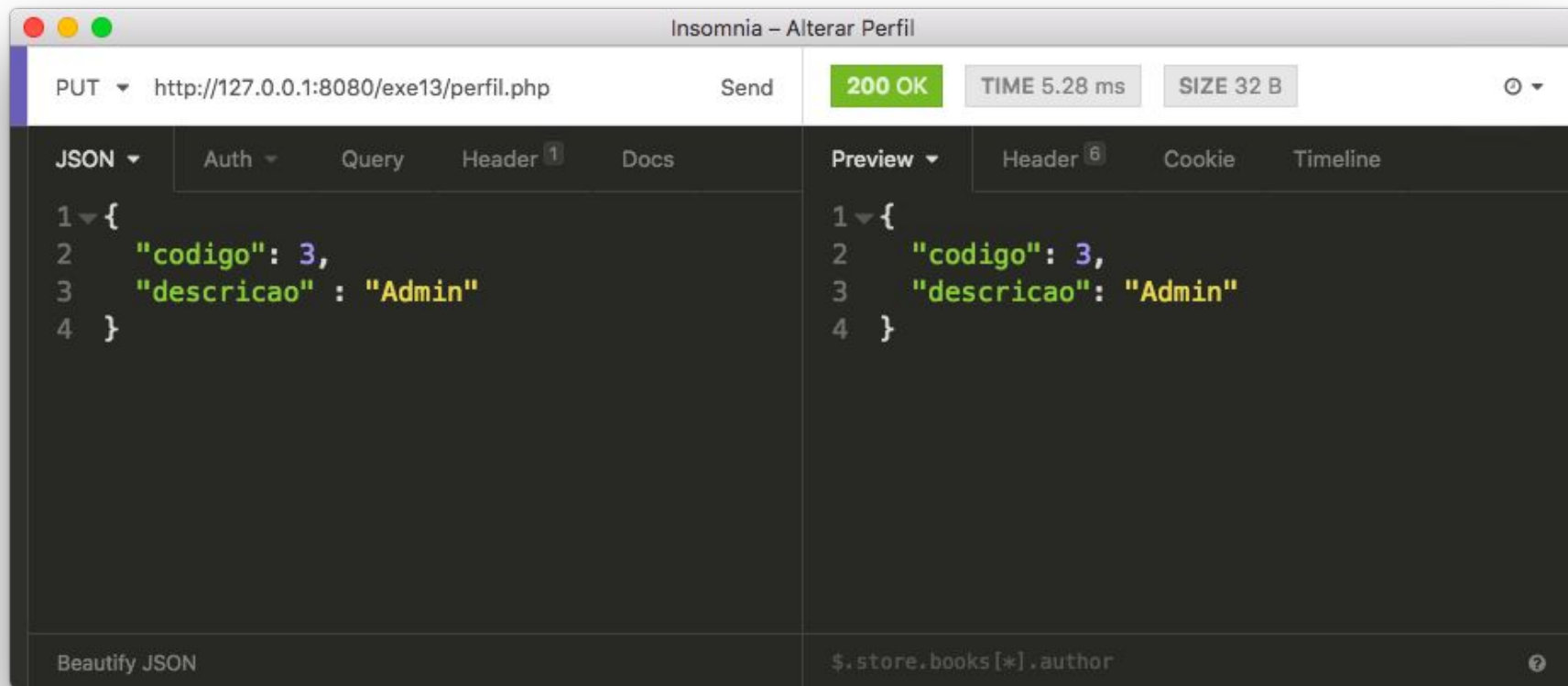
## Detalhes sobre a atualização de dados via API:

- Ao invés de **POST**, devemos utilizar o **PUT**.
- Em caso de sucesso, ao invés de retornar um *Status Code* **201**, devemos retornar o *Status Code* **200**.
- Os *Status Code* **405** (método não permitido) e **400** (requisição mal feita) podem ser utilizados de forma idêntica ao POST.
- Também é uma boa prática retornar o objeto alterado como resposta ao próprio método **PUT**.

**Atenção:** o método **PATCH** também pode ser utilizado para alterar recursos. Neste caso, porém, recomenda-se utilizá-lo apenas quando vamos alterar **uma parte** do recurso. Alterações completas do recurso devem ser feitas pelo método **PUT**.

# Atualizando dados pela API (exe13/perfil.php)

Veja a atualização dos dados pelo Insomnia REST.



# Apagando dados pela API (exe14/perfil.php)

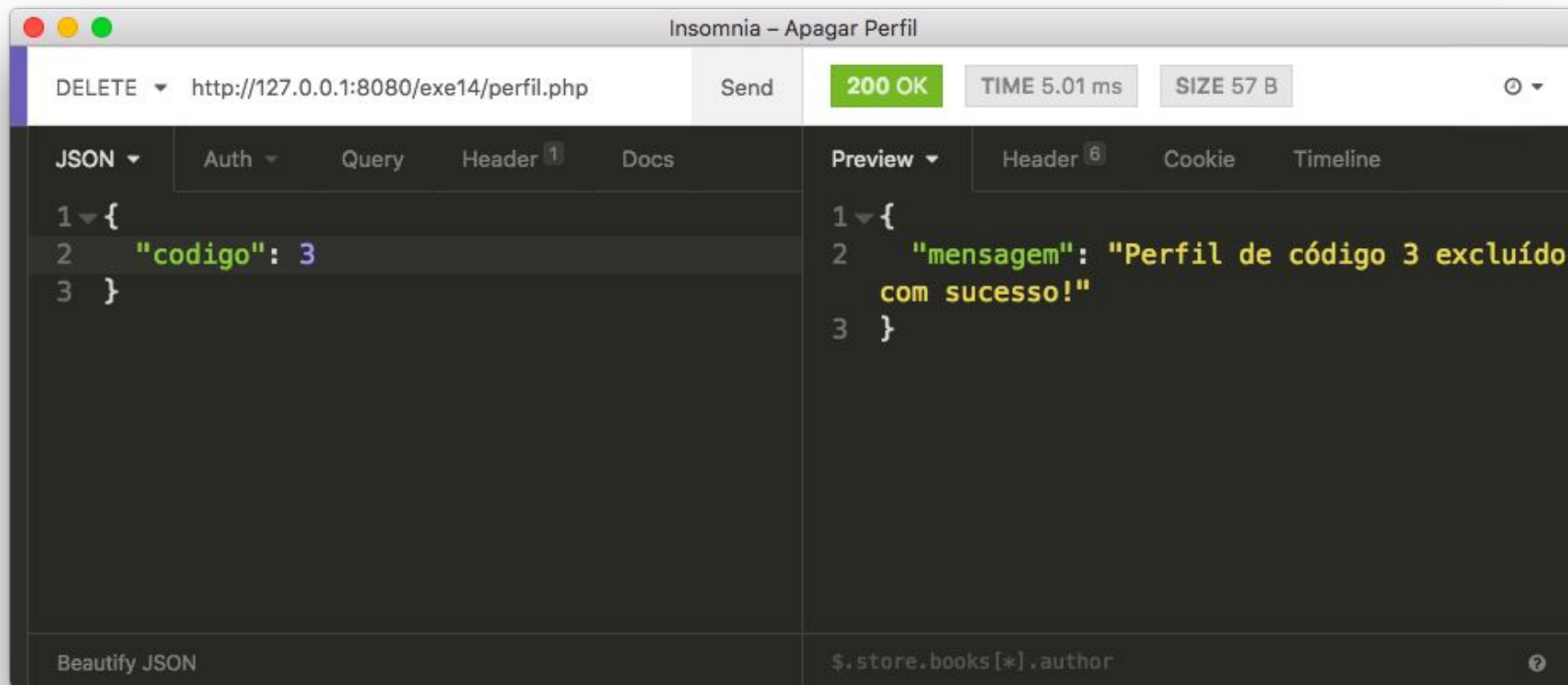
A exclusão de dados via API é semelhante à inserção. Veja as diferenças:

## Detalhes sobre a exclusão de dados via API:

- Ao invés de **POST**, devemos utilizar o **DELETE**.
- Em caso de sucesso, ao invés de retornar um *Status Code* **201**, devemos retornar o *Status Code* **200**. Neste caso, ao invés de retornar o objeto, podemos simplesmente retornar uma mensagem de sucesso informando o código do registro excluído.
- Em caso de falha (registro não encontrado), devemos retornar um *Status Code* **404** com uma mensagem informando o código do registro que não foi encontrado.
- Os *Status Code* **405** (método não permitido) e **400** (requisição mal feita) podem ser utilizados de forma idêntica ao POST.

# Apagando dados pela API (exe14/perfil.php)

Veja a exclusão dos dados pelo Insomnia REST.



# Retornando API a partir de classes



# Orientação a objetos em APIs (exe15.php)

Conversamos um pouco sobre como converter classes (ou melhor, suas instâncias, os objetos) em formato JSON para retornar pela API.

```
class Cliente {  
    public $nome;  
    public $idade;  
    public $aposentado;  
    public function __construct($nome,$idade,$aposentado) {  
        $this->nome = $nome;  
        $this->idade = $idade;  
        $this->aposentado = $aposentado;  
    }  
}  
  
$cliente = new Cliente("João",40,false);  
  
header('Content-Type: application/json');  
echo json_encode($cliente, JSON_UNESCAPED_UNICODE);  
die();
```



Note que todos os atributos da classe *Cliente* estão definidos como públicos.

**Por que?**

Isso acontece porque a função *json\_encode()* só consegue converter atributos públicos.

# Orientação a objetos em APIs (exe16.php)

```
class Cliente implements JsonSerializable {  
    private $nome;  
    private $idade;  
    private $aposentado;  
  
    public function __construct($nome,$idade,$aposentado) {  
        $this->nome = $nome;  
        $this->idade = $idade;  
        $this->aposentado = $aposentado;  
    }  
  
    public function jsonSerialize() {  
        $vars = get_object_vars($this);  
        return $vars;  
    }  
}
```



## Mas há uma alternativa!

Se implementarmos a interface *JsonSerializable* e por consequência codificarmos o método *jsonSerialize()*, a função *json\_encode()* saberá como converter o objeto, mesmo se todos seus atributos forem privados.

Obrigado!