

SW-II

SISTEMAS WEB II

Prof. Anderson Vanin

AULA 06 – APIs e Microserviços

APIs e Microserviços

O que é uma API?

API (Application Programming Interface) é um conjunto de regras e definições que permitem a comunicação entre diferentes sistemas. Uma **API** define como os sistemas podem interagir, quais **métodos podem ser chamados**, **quais parâmetros** devem ser usados e qual o **formato da resposta**.

Tipos de APIs

1. **APIs Locais:** Usadas dentro de um mesmo software.
2. **APIs Web:** Permitem a comunicação entre sistemas através da internet.
3. **APIs de Bibliotecas:** Disponibilizam funções de uma biblioteca para outros softwares.
4. **APIs de Sistema Operacional:** Permitem interação com o SO, como as APIs do Windows ou Linux.

Como funciona uma API Web?

Uma API Web geralmente segue o padrão **REST** (*Representational State Transfer*), que é baseado em **requisições HTTP**. Os principais métodos são:

- **GET**: Recupera informações.
- **POST**: Envia dados para criação de um novo recurso.
- **PUT**: Atualiza um recurso existente.
- **DELETE**: Remove um recurso.

Exemplo de uma API REST

Uma API que gerencia usuários pode ter os seguintes endpoints:

- **GET** **/usuarios** - Retorna a lista de usuários.
- **GET** **/usuarios/{id}** - Retorna um usuário específico.
- **POST** **/usuarios** - Cria um novo usuário.
- **PUT** **/usuarios/{id}** - Atualiza um usuário.
- **DELETE** **/usuarios/{id}** - Remove um usuário.

Método HTTP

URL (REQUISIÇÃO)

FUNÇÃO

Exemplo de uma API REST

A resposta normalmente é em formato JSON:

```
{  
  "id": 1,  
  "nome": "João Silva",  
  "email": joao@email.com  
}
```

O que são Microserviços?

Microserviços são uma abordagem arquitetural onde uma aplicação é dividida em **módulos independentes e menores**, chamados "serviços". **Cada serviço é responsável por uma funcionalidade específica e se comunica com outros serviços por meio de APIs.**

Benefícios dos Microserviços

- **Escalabilidade:** Cada serviço pode ser escalado independentemente.
- **Facilidade de manutenção:** Alterar um serviço não afeta toda a aplicação.
- **Resiliência:** Falhas em um serviço não derrubam a aplicação inteira.
- **Flexibilidade tecnológica:** Cada serviço pode ser desenvolvido com uma tecnologia diferente.

Diferença entre Monolitos e Microserviços

Monolito	<u>Microserviços</u>
Toda a lógica em um único código	Separado em pequenos serviços
Difícil de escalar	Facilmente escalável
Difícil de manter	Mais fácil de modificar
Uma falha pode afetar tudo	Falhas são isoladas

Exemplo de Arquitetura de Microserviços

Em um sistema de e-commerce, poderíamos ter os seguintes microserviços:

- **Usuários:** Gerencia cadastro e login.
- **Produtos:** Gerencia catálogo de produtos.
- **Pedidos:** Processa compras.
- **Pagamentos:** Processa transações financeiras.

Cada um desses serviços se comunica via **APIs REST**.

Requisições HTTP em APIs

Uma API Web geralmente utiliza o protocolo HTTP (HyperText Transfer Protocol) para comunicação. Esse protocolo permite que clientes (navegadores, aplicativos, sistemas) enviem requisições para servidores e recebam respostas.

Componentes de uma Requisição HTTP

Uma requisição HTTP tem os seguintes elementos principais:

1. **URL (Uniform Resource Locator)**

- Especifica o endereço do recurso que queremos acessar.
- Exemplo: <https://api.meusite.com/usuarios/1>

2. **Método HTTP**

- Define a ação que será realizada no recurso.
- Métodos mais usados:
 - **GET**: Obtém dados.
 - **POST**: Envia dados para criação.
 - **PUT**: Atualiza um recurso existente.
 - **DELETE**: Remove um recurso.

Componentes de uma Requisição HTTP

3. Cabeçalhos (Headers)

- Contêm informações adicionais, como autenticação e formato da resposta.
- Exemplos:

Content-Type: application/json

Authorization: Bearer token_de_acesso

Componentes de uma Requisição HTTP

3. Corpo da Requisição (Body)

- Usado em métodos como **POST** e **PUT** para enviar dados no formato *JSON* ou *XML*.
- Exemplo de um corpo *JSON* para criar um usuário:

```
{  
  "nome": "Maria Souza",  
  "email": "maria@email.com",  
  "senha": "123456"  
}
```

Resposta HTTP

Após uma requisição, o **servidor envia uma resposta HTTP**, que contém:

1. **Código de Status HTTP:** Indica o resultado da requisição. Alguns exemplos:
 - **200 OK** → Requisição bem-sucedida.
 - **201 Created** → Recurso criado com sucesso.
 - **400 Bad Request** → Erro na requisição (dados inválidos).
 - **401 Unauthorized** → Falta de autenticação.
 - **404 Not Found** → Recurso não encontrado.
 - **500 Internal Server Error** → Erro no servidor.

Resposta HTTP

2. **Cabeçalhos da Resposta:** Contêm informações sobre a resposta, como tipo de conteúdo:

Content-Type: application/json

Resposta HTTP

3. **Corpo da Resposta:** Normalmente retorna um JSON com os dados solicitados. Exemplo de resposta de um **GET** /usuarios/1:

```
{  
  "id": 1,  
  "nome": "Maria Souza",  
  "email": "maria@email.com"  
}
```

Criando uma API Simples em PHP (Servidor HTTP)

Passo 1: Criando um endpoint para listar usuários

Criamos um arquivo **api.php** que responde a um **GET** em **/usuarios**.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'AULA06' containing a file named 'api.php'. The code editor shows the contents of 'api.php' with the following code:

```
1  <?php
2      //CABEÇALHO
3      header("Content-Type: application/json"); // Define o tipo de resposta
4
5      //CONTEÚDO
6      $usuarios = [
7          ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
8          ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
9      ];
10
11     // Converte para JSON e retorna
12     echo json_encode($usuarios);
13
14  ?>
```

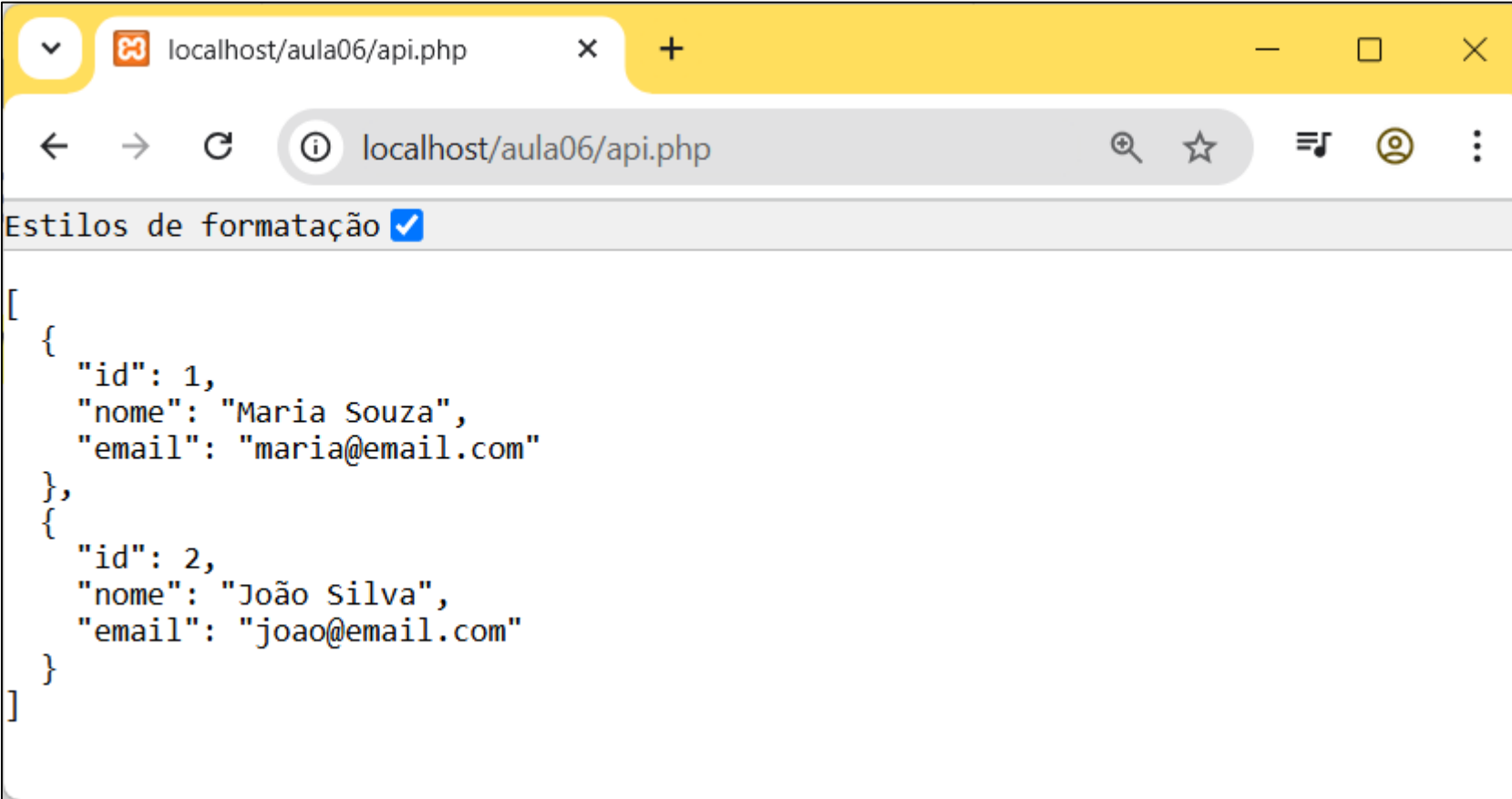
Passo 1: Criando um endpoint para listar usuários

Agora, quando acessarmos **http://localhost/aulao6/api.php** no navegador receberemos:



Passo 1: Criando um endpoint para listar usuários

<http://localhost/aulao6/api.php>



```
[
  {
    "id": 1,
    "nome": "Maria Souza",
    "email": "maria@email.com"
  },
  {
    "id": 2,
    "nome": "João Silva",
    "email": "joao@email.com"
  }
]
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Agora vamos modificar nossa API para que ela possa aceitar requisições **POST** e adicionar um novo usuário.

IMPORTANTE: Quando digitamos uma URL no navegador e teclamos enter, o método da requisição solicitado é do tipo **GET**. Para enviarmos um dado ou conjunto de dados precisamos enviar utilizando o método **POST**.

Altere o código do arquivo **api.php** de modo a podermos verificar qual método está sendo solicitado em um determinado *endpoint* (url).

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

EXPLORER

▼ AULA06

api.php

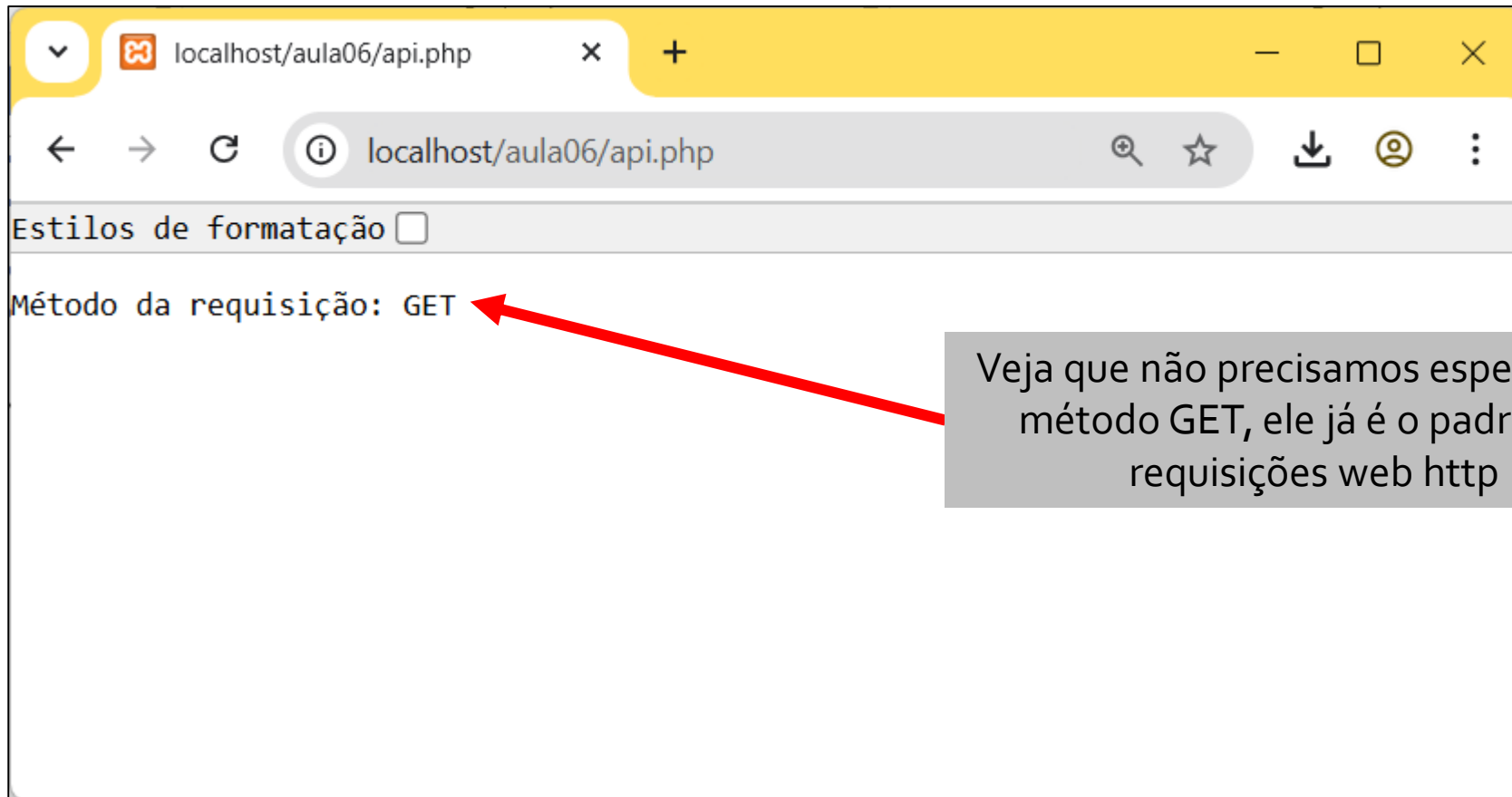
Deixe o restante do código comentado

api.php

```
1  <?php
2  //CABEÇALHO
3  header("Content-Type: application/json"); // Define o tipo de resposta
4
5  $metodo = $_SERVER['REQUEST_METHOD'];
6
7  echo "Método da requisição: ".$metodo;
8
9  //CONTEÚDO
10 // $usuarios = [
11 //     ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
12 //     ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
13 // ];
14
15 // Converte para JSON e retorna
16 //echo json_encode($usuarios);
17
18 ?>
```


Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Digite a mesma url em seu navegador: <http://localhost/aulao6/api.php>



Veja que não precisamos especificar o método GET, ele já é o padrão de requisições web http

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

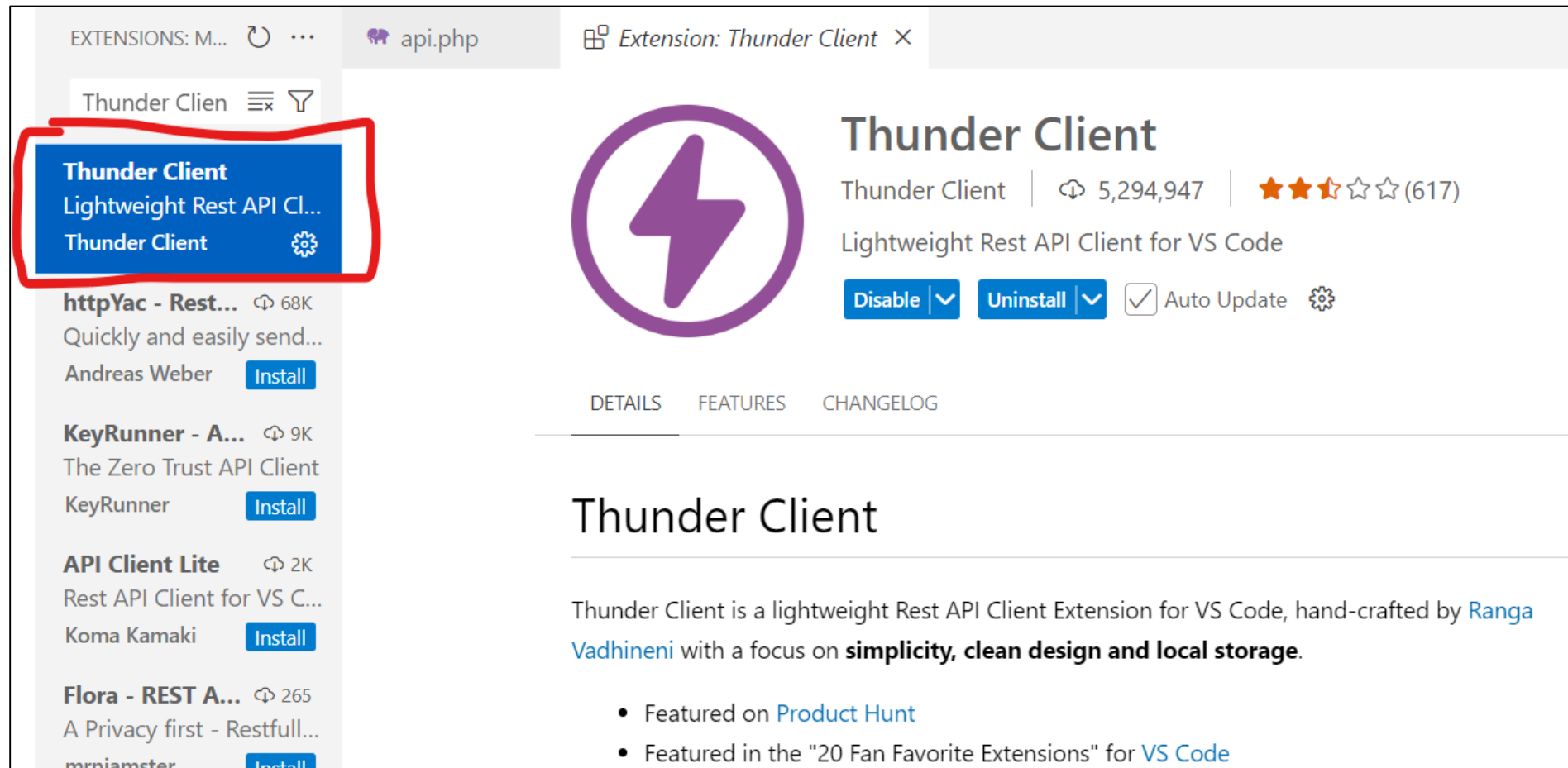
Para enviar requisições com outros métodos diferentes de GET, podemos fazer isso por meio de código enviando no corpo da requisição qual o método desejado. Mas só para testarmos vamos utilizar um plugin do vscode, no qual podemos especificar várias opções e tipos de requisições à APIs, além de verificar o retorno destas requisições.

No VSCODE clique em extensões:



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Procure pela extensão Thunder Client e instale-a.



The screenshot shows the VS Code interface with the Extensions view open. The search bar contains 'Thunder Client'. The extension 'Thunder Client' by Ranga Vadhineni is highlighted with a red box. The extension details panel on the right shows the extension's icon (a purple lightning bolt), name, version (5.294.947), and a 4-star rating. It also shows buttons for 'Disable', 'Uninstall', and 'Auto Update'. The description states it is a lightweight Rest API Client for VS Code, hand-crafted by Ranga Vadhineni, focusing on simplicity, clean design, and local storage. It is also mentioned as being featured on Product Hunt and in the '20 Fan Favorite Extensions' for VS Code.

EXTENSIONS: M... ... api.php Extension: Thunder Client

Thunder Clie...

Thunder Client
Lightweight Rest API Cl...
Thunder Client

httpYac - Rest... 68K
Quickly and easily send...
Andreas Weber

KeyRunner - A... 9K
The Zero Trust API Client
KeyRunner

API Client Lite 2K
Rest API Client for VS C...
Koma Kamaki

Flora - REST A... 265
A Privacy first - Restfull...
mrniaster

Thunder Client
Thunder Client | 5,294,947 | (617)
Lightweight Rest API Client for VS Code
 ☒ Auto Update

DETAILS FEATURES CHANGELOG

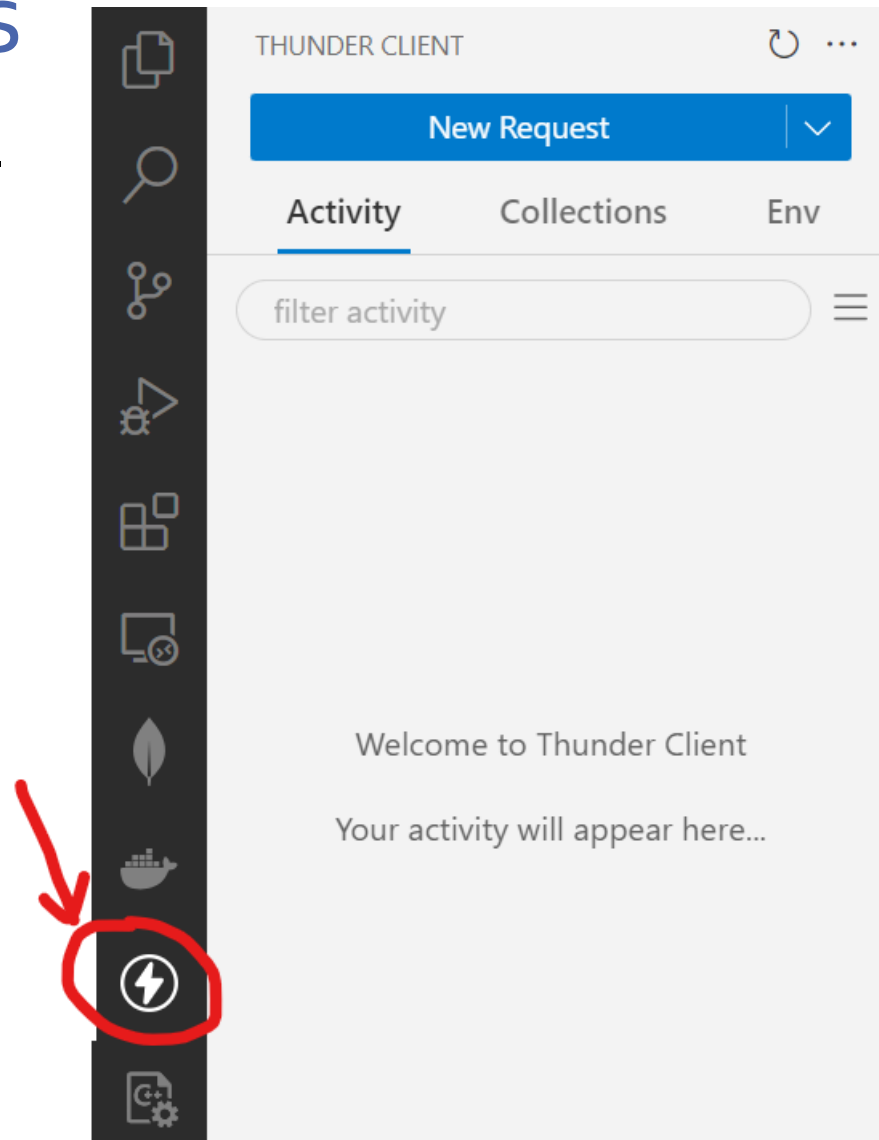
Thunder Client

Thunder Client is a lightweight Rest API Client Extension for VS Code, hand-crafted by [Ranga Vadhineni](#) with a focus on **simplicity, clean design and local storage**.

- Featured on [Product Hunt](#)
- Featured in the "20 Fan Favorite Extensions" for [VS Code](#)

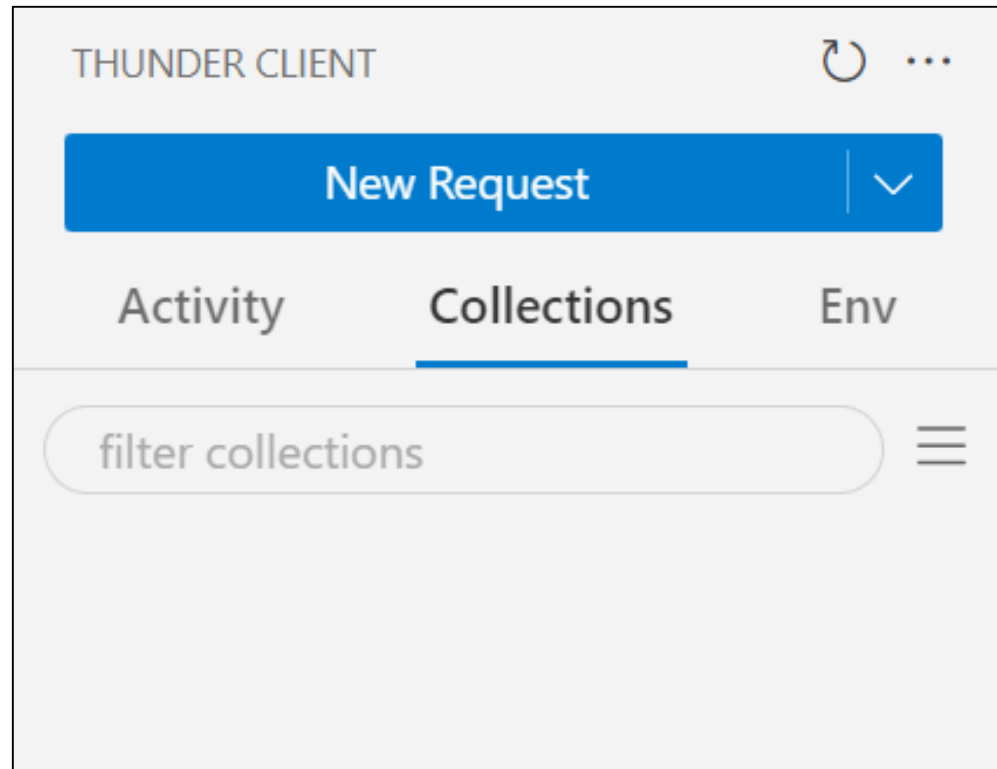
Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Clique para abrir Thunder Client.



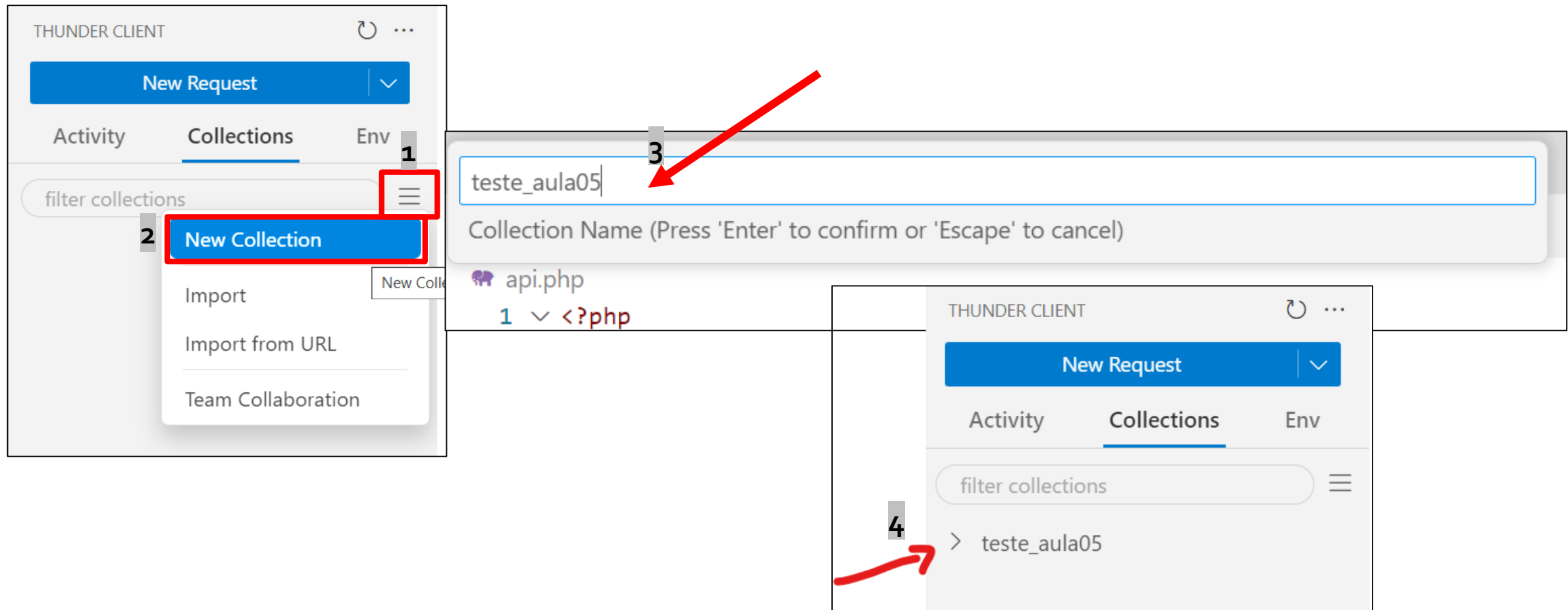
Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Clique em Collections. Uma Collection irá conter as rotas (endpoints, métodos e parâmetros de cada tipo de requisição que podemos fazer em APIs).



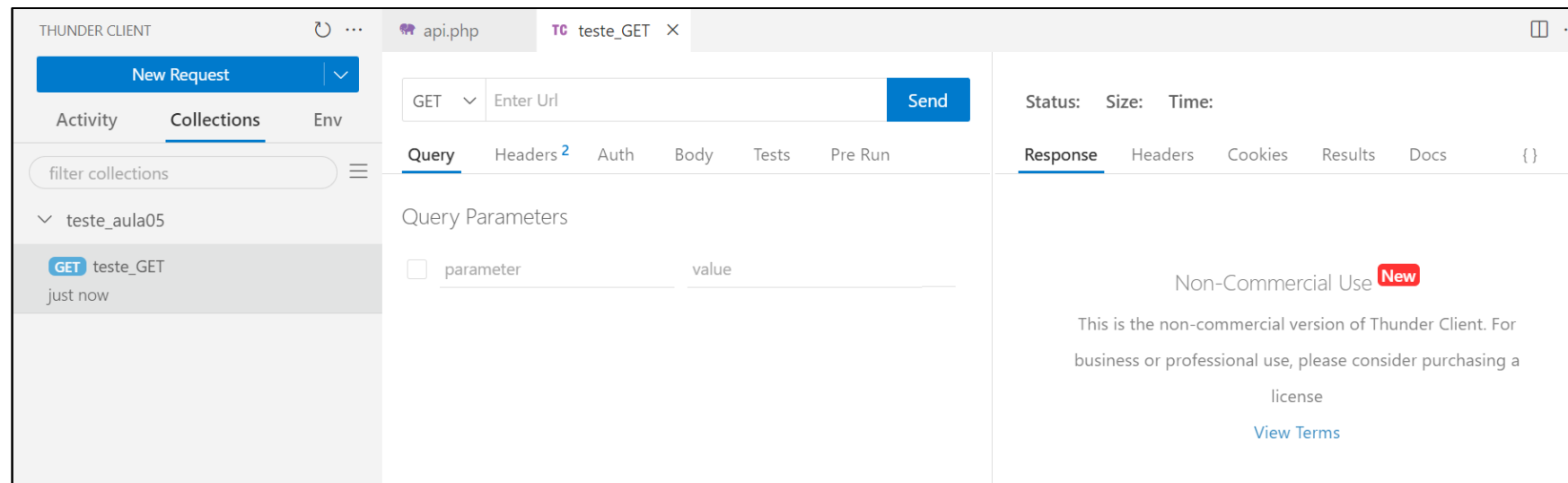
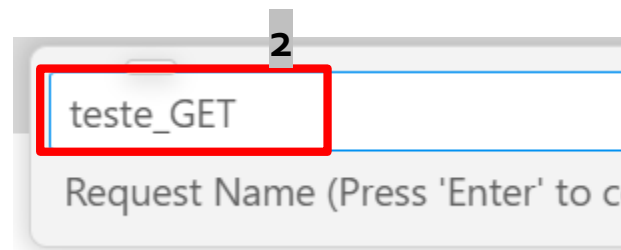
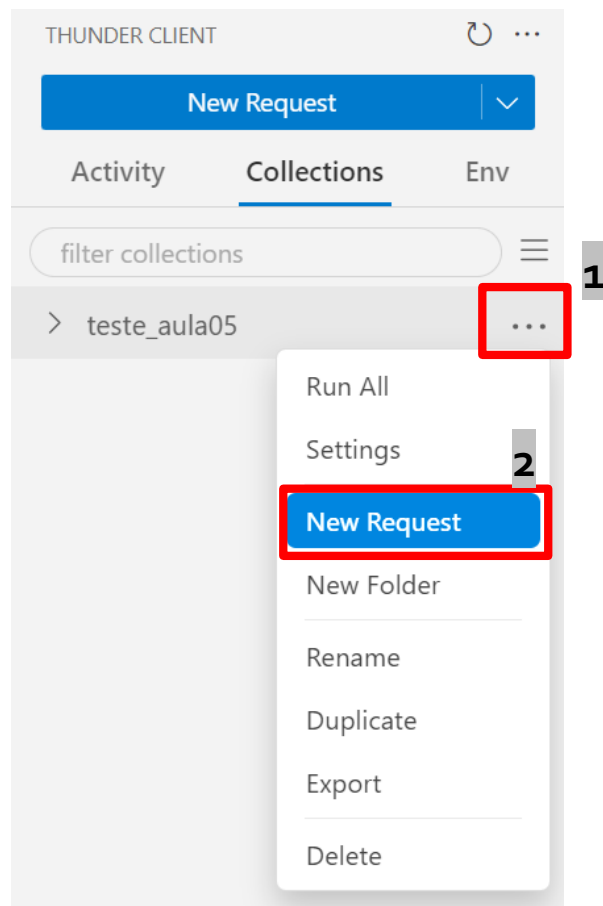
Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Adicione uma nova Collections e dê um nome de: **teste_aula05**



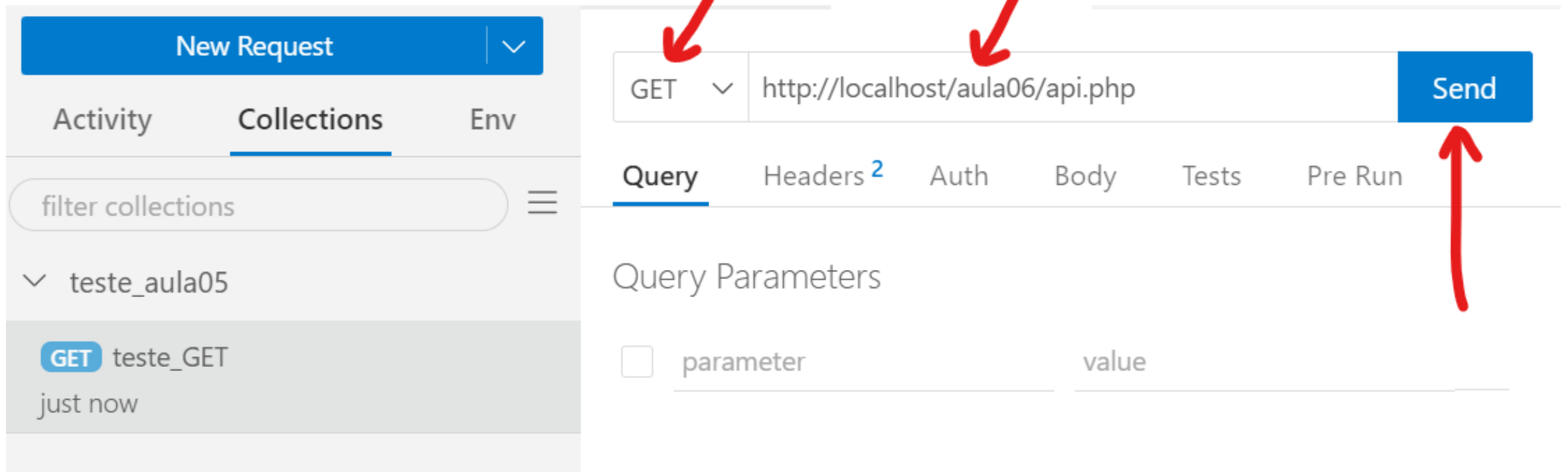
Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Agora vamos criar as requisições. Crie uma requisição do tipo **GET** com o nome de **teste_GET**



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Vamos configurar essa requisição para **GET** na url <http://localhost/aulao6/api.php>. Em seguida clique em SEND para enviar a requisição e observe a resposta.



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Resposta da requisição para **GET** na url <http://localhost/aulao6/api.php>.

The screenshot shows a web client interface with a tab labeled 'api.php' and a test named 'teste_GET'. The request is configured with the method 'GET' and the URL 'http://localhost/aula06/api.php'. The 'Send' button is highlighted with a red box. Below the request configuration, the 'Query' tab is selected, showing a table for 'Query Parameters' with columns 'parameter' and 'value'. The response section shows a status of '200 OK', a size of '28 Bytes', and a time of '9 ms'. The 'Response' tab is selected, showing a single entry: '1 Método da requisição: GET'. Red arrows point to the '200 OK' status and the response text.

api.php teste_GET X

GET Send

Query Headers² Auth Body Tests Pre Run

Query Parameters

	parameter	value
--	-----------	-------

Status: 200 OK Size: 28 Bytes Time: 9 ms

Response Headers⁶ Cookies Results Docs {} ≡

1 Método da requisição: GET

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Vamos criar um requisição com método **POST** na url <http://localhost/aulao6/api.php>.

The image shows a sequence of steps to create a new POST request in Thunder Client:

1. A red box highlights the three-dot menu icon in the top right corner of the 'teste_aula05' collection.
2. A red box highlights the 'New Request' button in the dropdown menu.
3. A red box highlights the input field where the request name 'teste_POST' is being typed.
4. A red box highlights the 'Send' button in the request configuration panel.

The main interface of Thunder Client is visible, showing the 'Collections' tab with the 'teste_aula05' collection. The 'teste_POST' request is listed at the bottom. The 'Query' tab is selected in the request configuration panel, and the URL 'http://localhost/aulao6/api.php' is entered. The 'Response' tab shows the status '200 OK' and the message 'Método da requisição: POST'.

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Agora podemos criar tipos diferentes de requisições alterando o tipo de método utilizando uma determinada url, ou seja, num mesmo endpoint, podemos ter tipos diferentes de requisições mudando apenas o método.

Agora que podemos inferir um método, vamos para a lógica do arquivo que recebe a requisição, e aí podemos, por exemplo, utilizar uma estrutura do tipo **switch case** para avaliar qual método foi requisitado e definir as ações que devem ser executadas.


Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Agora podemos criar tipos diferentes de requisições alterando o tipo de método utilizando uma determinada url, ou seja, num mesmo endpoint, podemos ter tipos diferentes de requisições mudando apenas o método.

Agora que podemos inferir um método, vamos para a lógica do arquivo que recebe a requisição, e aí podemos, por exemplo, utilizar uma estrutura do tipo **switch case** para avaliar qual método foi requisitado e definir as ações que devem ser executadas.

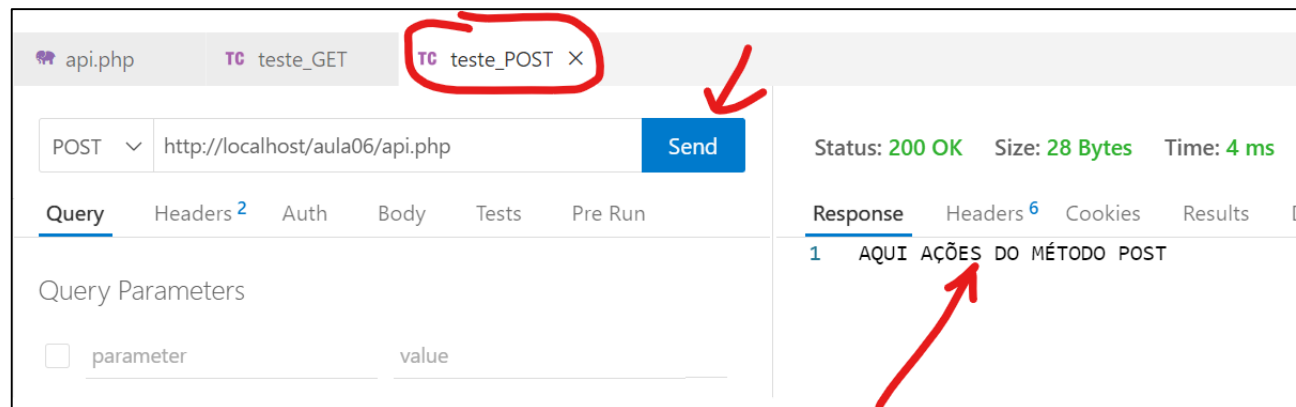
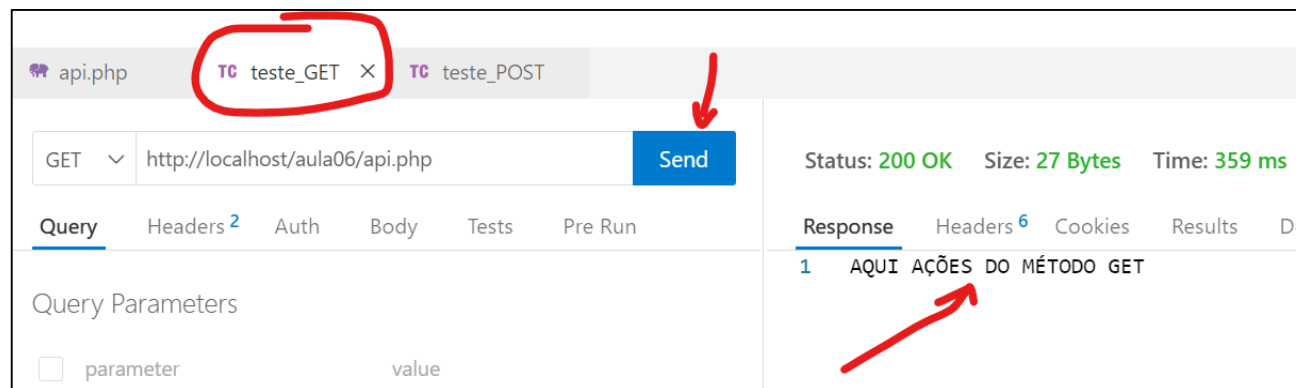
Passo 2: Criando um endpoint que aceita POST para adicionar usuários

```
api.php
1  <?php
2      //CABEÇALHO
3      header("Content-Type: application/json"); // Define o tipo de resposta
4
5      $metodo = $_SERVER['REQUEST_METHOD'];
6
7      //echo "Método da requisição: ".$metodo;
8
9      switch ($metodo) {
10         case 'GET':
11             echo "AQUI AÇÕES DO MÉTODO GET";
12             break;
13         case 'POST':
14             echo "AQUI AÇÕES DO MÉTODO POST";
15             break;
16         default:
17             echo "MÉTODO NÃO ENCONTRADO!";
18             break;
19     }
20
21
22     //CONTEÚDO
23     // $usuarios = [
```



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Teste as rotas com uso do Thunder Client e as requests criadas anteriormente. Para isso basta clicar em SEND novamente em cada uma das requests.



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Para efeito de testes, vamos deixar o array usado anteriormente definido no início da nossa api como dados fictícios iniciais.

```
api.php
1  <?php
2      //CABEÇALHO
3      header("Content-Type: application/json"); // Define o tipo de resposta
4
5      $metodo = $_SERVER['REQUEST_METHOD'];
6
7      //CONTEÚDO
8      $usuarios = [
9          ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
10         ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
11     ];
12
13     switch ($metodo) {
14         case 'GET':
15             echo "AQUI AÇÕES DO MÉTODO GET";
16             break;
17         case 'POST':
18             echo "AQUI AÇÕES DO MÉTODO POST";
19             break;
20         default:
21             echo "MÉTODO NÃO ENCONTRADO!";
22             break;
23     }
24
25
26     // Converte para JSON e retorna
27     //echo json_encode($usuarios);
28
29     ?>
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

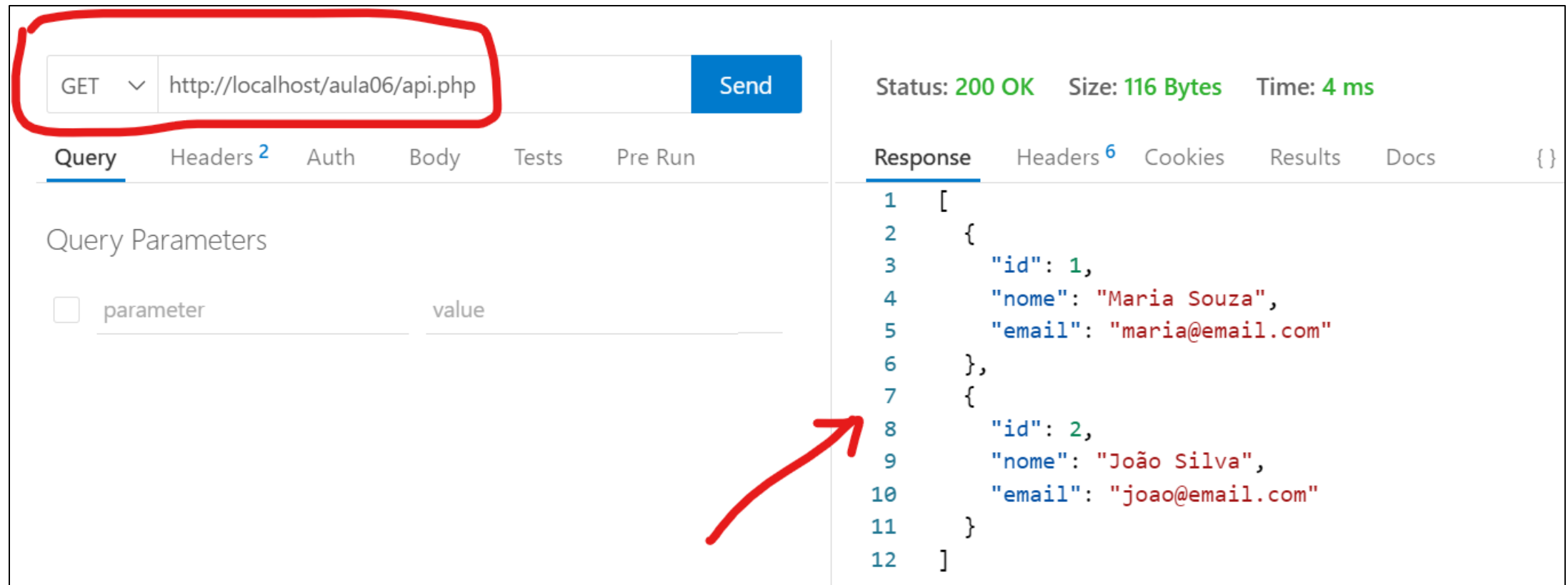
Para o método GET, precisamos somente exibir o conteúdo de \$usuários em formato JSON.

```
api.php
1  <?php
2  //CABEÇALHO
3  header("Content-Type: application/json"); // Define o tipo de resposta
4  $metodo = $_SERVER['REQUEST_METHOD'];
5
6  //CONTEÚDO
7  $usuarios = [
8      ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
9      ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
10 ];
11
12 switch ($metodo) {
13     case 'GET':
14         //echo "AQUI AÇÕES DO MÉTODO GET";
15         // Converte para JSON e retorna
16         echo json_encode($usuarios);
17         break;
18     case 'POST':
19         echo "AQUI AÇÕES DO MÉTODO POST";
20         break;
21     default:
22         echo "MÉTODO NÃO ENCONTRADO!";
23         break;
24 }
25 ?>
```



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Teste no thunder client



The screenshot displays the Thunder Client interface. At the top, a red rectangle highlights the request configuration: the method is set to 'GET' and the URL is 'http://localhost/aula06/api.php'. A blue 'Send' button is located to the right of the URL. Below this, the 'Query' tab is selected, showing a table for 'Query Parameters' with columns for 'parameter' and 'value'. The 'Response' tab is also visible, showing a JSON array of two user objects. A red arrow points from the 'Send' button area to the first user object in the response.

GET ⌵ http://localhost/aula06/api.php Send

Status: 200 OK Size: 116 Bytes Time: 4 ms

Query Parameters

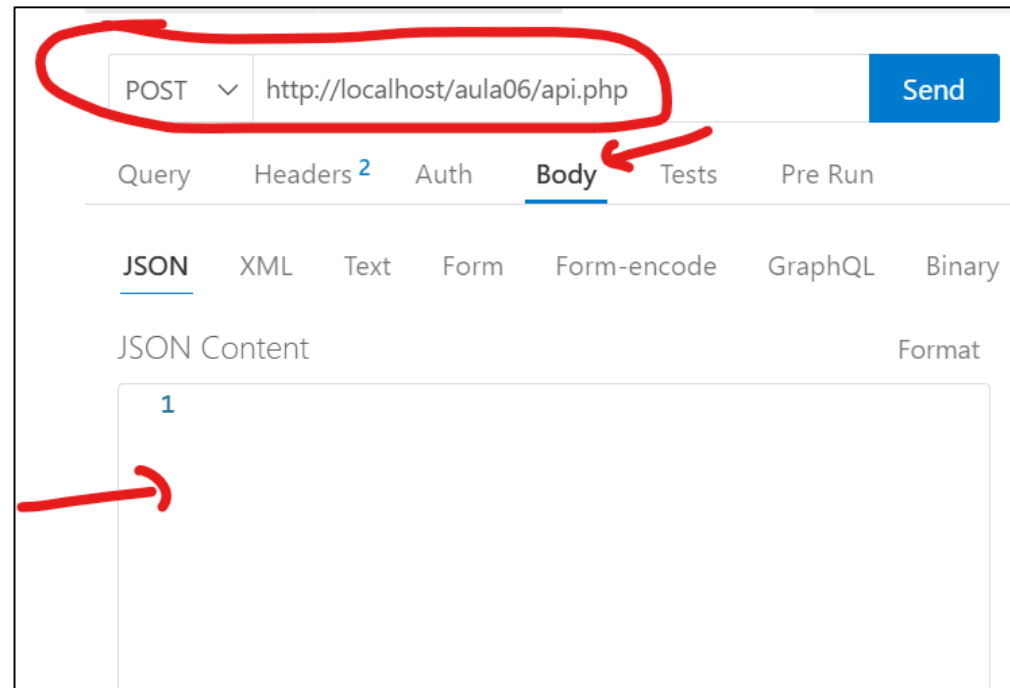
parameter	value
<input type="checkbox"/>	

Response

```
1  [  
2    {  
3      "id": 1,  
4      "nome": "Maria Souza",  
5      "email": "maria@email.com"  
6    },  
7    {  
8      "id": 2,  
9      "nome": "João Silva",  
10     "email": "joao@email.com"  
11   }  
12 ]
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Para adicionar um novo usuário à \$usuários, vamos precisar enviar os dados em formato JSON para a nossa API. Para isso vamos enviar estes dados no “corpo” da requisição. Abra a resquest para POST e clique em Body.



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Vamos digitar alguns dados fictícios em formato JSON (lembre-se: estes dados podem ser obtidos através de um formulário).

The screenshot shows a REST client interface with the following elements:

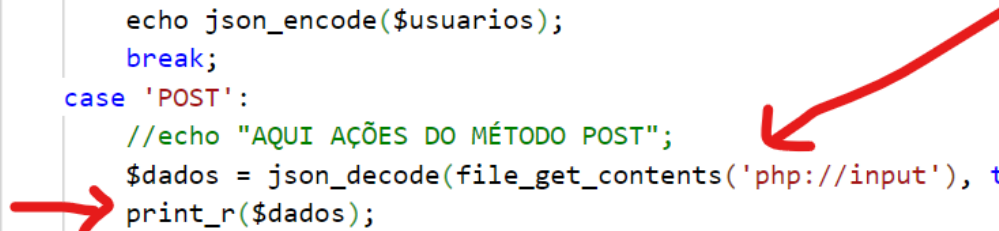
- Method and URL:** A dropdown menu set to "POST" and a text field containing "http://localhost/aula06/api.php". Both are circled in red.
- Send Button:** A blue button labeled "Send" is located to the right of the URL field.
- Body Tab:** The "Body" tab is selected and circled in red.
- JSON Tab:** The "JSON" sub-tab is selected and circled in red.
- JSON Content:** The JSON body is displayed in a text area with line numbers 1 through 5 on the left. A red arrow points to line 4. The content is:

```
1 {  
2   "id": 3,  
3   "nome": "Fulano Novo",  
4   "email": "fulano@email.com"  
5 }
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Agora precisamos ajustar a api, para que ela receba estes dados.

```
api.php x TC teste_GET TC teste_POST
api.php
1  <?php
2  //CABEÇALHO
3  header("Content-Type: application/json"); // Define o tipo de resposta
4  $metodo = $_SERVER['REQUEST_METHOD'];
5
6  //CONTEÚDO
7  $usuarios = [
8      ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
9      ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
10 ];
11
12 switch ($metodo) {
13     case 'GET':
14         //echo "AQUI AÇÕES DO MÉTODO GET";
15         // Converte para JSON e retorna
16         echo json_encode($usuarios);
17         break;
18     case 'POST':
19         //echo "AQUI AÇÕES DO MÉTODO POST";
20         $dados = json_decode(file_get_contents('php://input'), true);
21         print_r($dados);
22         break;
23     default:
24         echo "MÉTODO NÃO ENCONTRADO!";
25         break;
26 }
27 ?>
```



Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Vamos criar um array simples, que irá conter os dados que vieram da requisição POST, e depois nós podemos fazer uma adição deste array simples, no primeiro array inicial chamado \$usuários.

```
case 'POST':  
    //echo "AQUI AÇÕES DO MÉTODO POST";  
    $dados = json_decode(file_get_contents('php://input'), true);  
    //print_r($dados);  
    $novoUsuario = [  
        "id" => $dados["id"],  
        "nome" => $dados["nome"],  
        "email" => $dados["email"]  
    ];  
    break;
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Adicionando \$novoUsuario à \$usuarios.

```
case 'POST':  
    //echo "AQUI AÇÕES DO MÉTODO POST";  
    $dados = json_decode(file_get_contents('php://input'), true);  
    //print_r($dados);  
    $novoUsuario = [  
        "id" => $dados["id"],  
        "nome" => $dados["nome"],  
        "email" => $dados["email"]  
    ];  
  
    // Adiciona o novo usuário ao array existente  
    array_push($usuarios, $novoUsuario);  
    echo json_encode('Usuário inserido com sucesso!');  
    print_r($usuarios);  
  
    break;  
default:
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC.

The screenshot displays a REST client interface. The top bar shows the method 'POST' and the URL 'http://localhost/aula06/api.php'. A 'Send' button is located to the right. Below this, tabs for 'Query', 'Headers', 'Auth', 'Body', 'Tests', and 'Pre Run' are visible, with 'Body' selected. Under the 'Body' tab, sub-tabs for 'JSON', 'XML', 'Text', 'Form', 'Form-encode', 'GraphQL', and 'Binary' are shown, with 'JSON' selected. The 'JSON Content' field contains a JSON object:

```
{  "id": 3,  "nome": "Fulano Novo",  "email": "fulano@email.com"}
```

. To the right of the client, the response details are shown: 'Status: 200 OK', 'Size: 36 Bytes', and 'Time: 12 ms'. Below these, tabs for 'Response', 'Headers', 'Cookies', 'Results', and 'Docs' are visible, with 'Response' selected. The response body shows a single line:

```
1 "Usuário inserido com sucesso!"
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

Perceba que ao utilizarmos o método POST, é exibida a informação de que um novo usuário foi inserido, mas ao testar novamente o método GET você irá verificar que o resultado ainda permanece inalterado!

Vamos transformar os dados iniciais em um arquivo JSON e executar as manipulações neste arquivo.

- Crie o arquivo inicial em JSON e salve-o com o nome de **usuarios.json**

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

```
[  
  {  
    "id": 1,  
    "nome": "Maria Souza",  
    "email": "maria@email.com"  
  },  
  {  
    "id": 2,  
    "nome": "João Silva",  
    "email": "joao@email.com"  
  }  
]
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

▼ AULA06_SWII

🐘 api.php

{} usuarios.json

{} usuarios.json > ...

```
1  [  
2      {  
3          "id": 1,  
4          "nome": "Maria Souza",  
5          "email": "maria@email.com"  
6      },  
7      {  
8          "id": 2,  
9          "nome": "João Silva",  
10         "email": "joao@email.com"  
11     }  
12 ]
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

- Altere o código inicial da **api.php**

```
api.php
1  <?php
2  //CABEÇALHO
3  header("Content-Type: application/json; charset=UTF-8"); //DEFINE O TIPO DE RESPOSTA
4
5  $metodo = $_SERVER['REQUEST_METHOD'];
6  //echo "Método da requisição: " . $metodo;
7
8  // RECUPERA O ARQUIVO JSON NA MESMA PASTA DO PROJETO
9  $arquivo = 'usuarios.json';
10
11  // VERIFICA SE O ARQUIVO EXISTE, SE NÃO EXISTIR CRIA UM COM ARRAY VAZIO
12  if (!file_exists($arquivo)) {
13      file_put_contents($arquivo, json_encode([], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
14  }
15
16  // LÊ O CONTEÚDO DO ARQUIVO JSON
17  $usuarios = json_decode(file_get_contents($arquivo), true);
18
19  //CONTEÚDO
20  // $usuarios = [
21  //     ["id" => 1, "nome" => "Maria Souza", "email" => "maria@email.com"],
22  //     ["id" => 2, "nome" => "João Silva", "email" => "joao@email.com"]
23  // ];
24
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

- Altere o código inicial da **api.php**

```
25     switch ($metodo) {
26         case 'GET':
27             //echo "AQUI AS AÇÕES DO METODO GET";
28             //CONVERTE PARA JSON E RETORNA
29             echo json_encode($usuarios, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
30             break;
31         case 'POST':
32             //echo "AQUI AS AÇÕES DO METODO POST";
33             //LER OS DADOS NO CORPO DA REQUISIÇÃO
34             $dados = json_decode(file_get_contents('php://input'), true);
35             // print_r($dados);
36
37             // VERIFICA SE OS CAMPOS OBRIGATÓRIOS FORAM PREENCHIDOS
38             if (!isset($dados["id"]) || !isset($dados["nome"]) || !isset($dados["email"])) {
39                 http_response_code(400);
40                 echo json_encode(["erro" => "Dados incompletos."], JSON_UNESCAPED_UNICODE);
41                 exit;
42             }
43
44             // CRIA NOVO USUÁRIO
45             $novo_usuario = [
46                 "id" => $dados["id"],
47                 "nome" => $dados["nome"],
48                 "email" => $dados["email"],
49             ];
50     }
```

Passo 2: Criando um endpoint que aceita POST para adicionar usuários

- Altere o código inicial da **api.php**

```
50
51      // ADICIONA AO ARRAY DE USUÁRIOS
52      $usuarios[] = $novo_usuario;
53
54      // SALVA O ARRAY ATUALIZADO NO ARQUIVO JSON
55      file_put_contents($arquivo, json_encode($usuarios, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
56
57      // RETORNA MENSAGEM DE SUCESSO
58      echo json_encode(["mensagem" => "Usuário inserido com sucesso!", "usuarios" => $usuarios], JSON_UNESCAPED_UNICODE);
59      break;
60
61      //ADICIONA O NOVO USUARIO AO ARRAY EXISTENTE
62      // array_push($usuarios, $novo_usuario);
63      // echo json_encode('Usuário inserido com sucesso!', JSON_UNESCAPED_UNICODE);
64      // print_r($usuarios);
65
66      break;
67
68  default:
69      // echo "MÉTODO NÃO ENCONTRADO!";
70      // break;
71      http_response_code(405); // Método não permitido
72      echo json_encode(["erro" => "Método não permitido!"], JSON_UNESCAPED_UNICODE);
73      break;
74  }
75  ?>
```

Observações

- Ao adicionar usuários perceba que o **número de ID não é alterado!** **Teremos dados duplicados!**
Precisamos pensar em uma lógica para isso também.
- Precisamos criar mais alguns ***endpoints*** como por exemplo: listar um cliente pelo seu id, atualizar um cliente pelo id, apagar um cliente pelo id, etc...

NAS PRÓXIMAS AULAS IREMOS TRATAR DESSES ITENS.