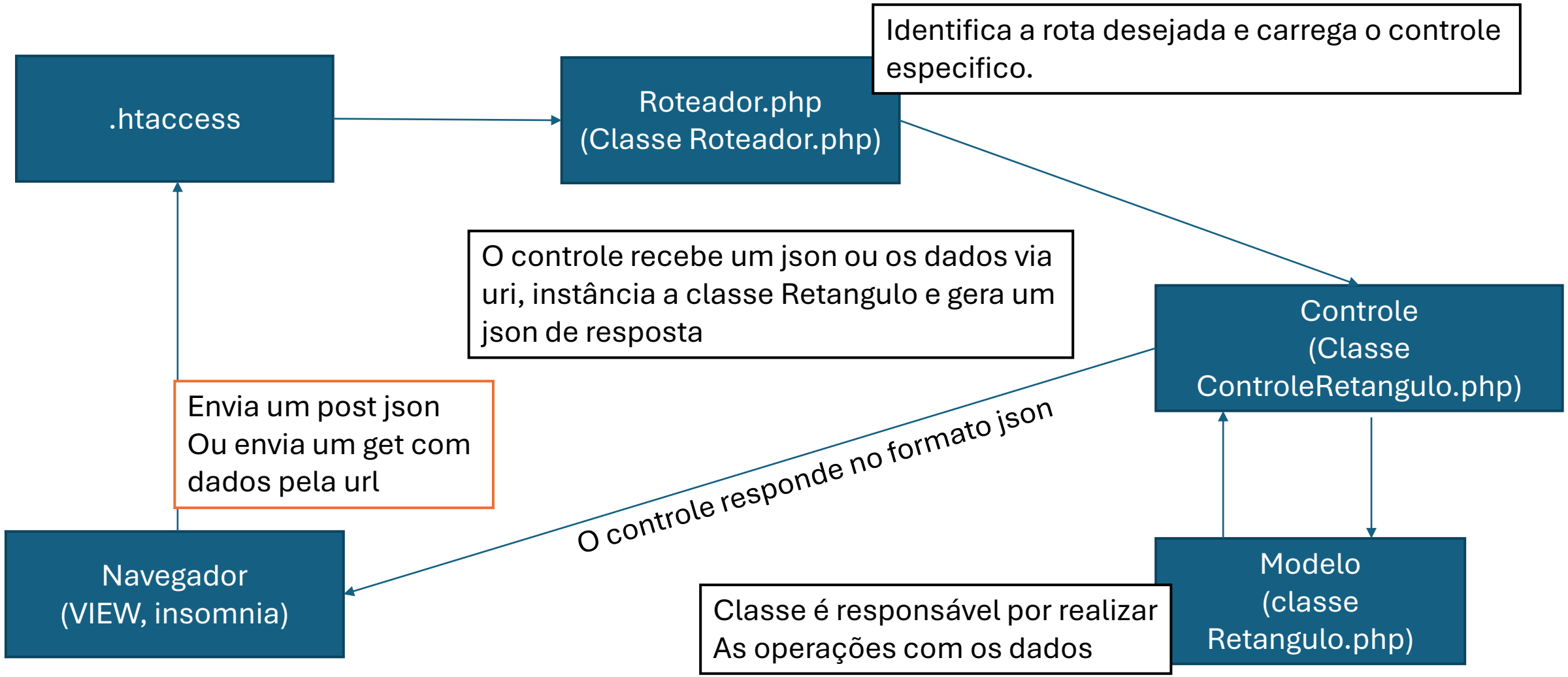
Minha primeira API: API Retângulo

Prof. Me. Hélio Esperidião

Exemplo 1: API Retangulo

- Será apresentada a criação de uma API no Modelo MVC.
- O front-end envia um GET com a URL no seguinte padrão:
  - http://localhost/retangulos/serviço/variavel1/variavel2
    - http://localhost/retangulos/area/3/4
    - http://localhost/retangulos/diagonal/3/4
    - http://localhost/retangulos/perimetro/3/4
    - http://localhost/retangulos/3/4
- A API recebe do cliente dados e responde um JSON.

Arquitetura da API

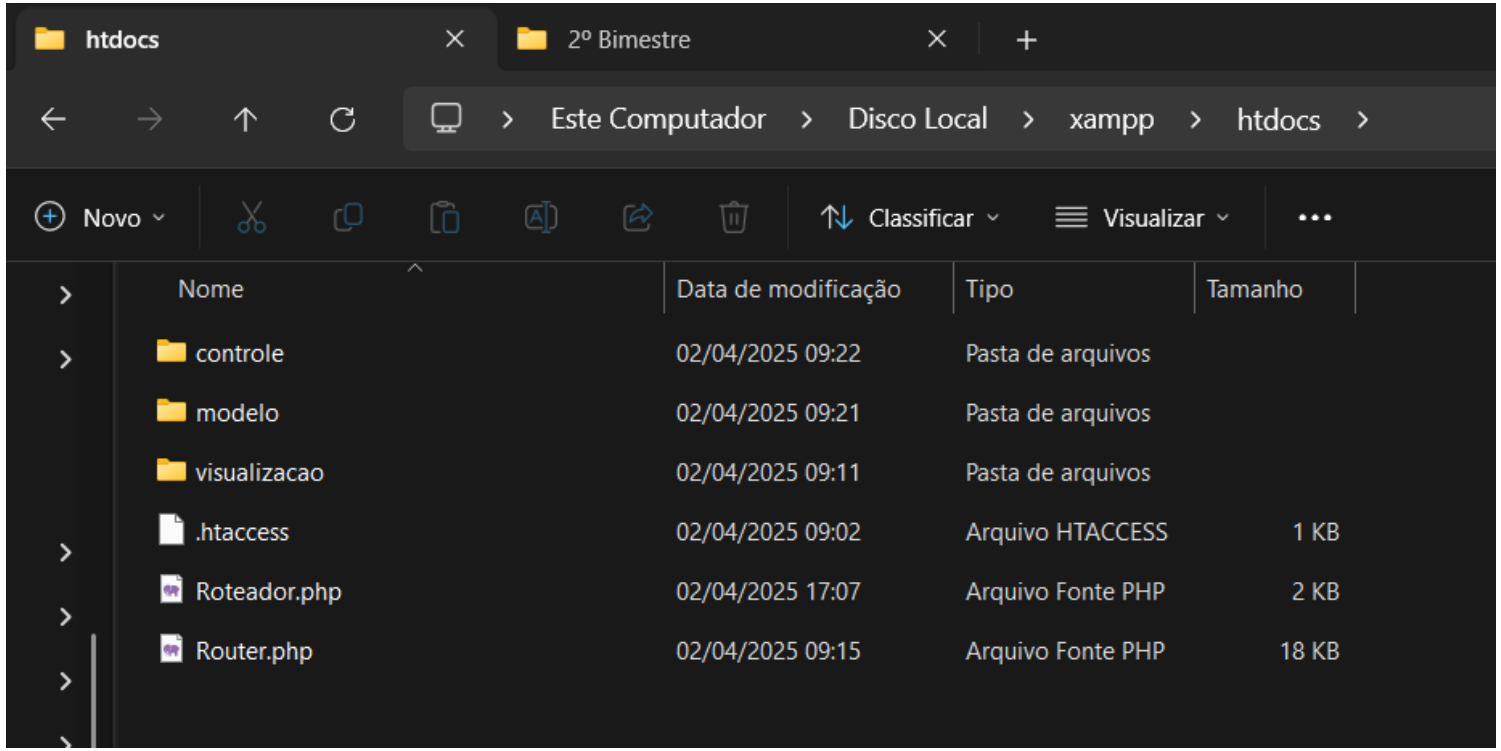


Exemplo: Arquivos

- Arquivos pasta raiz: \htdocs\
- Configuração e roteamento
  - .htaccess
  - Roteador.php
  - Router.php
- Pasta: controle
  - ControleRetangulo.php
- Pasta: modelo
  - Retangulo.php

Estrutura de Arquivos

- Para fins didáticos apague todos os arquivos da pasta htdocs e posicione todos os arquivos no diretório: C:\xampp\htdocs
- CRIE A ESTRUTURA:

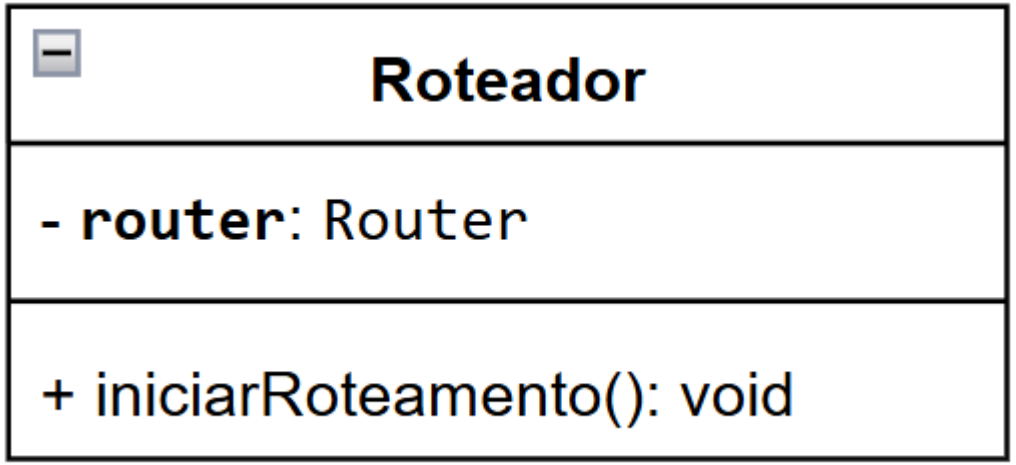


Lógica das classes

- Esquema:
  - O Roteador identifica a rota.
    - O roteador utiliza os métodos da Classe ControleRetangulo
      - ControleRetangulo utiliza a classe Retangulo

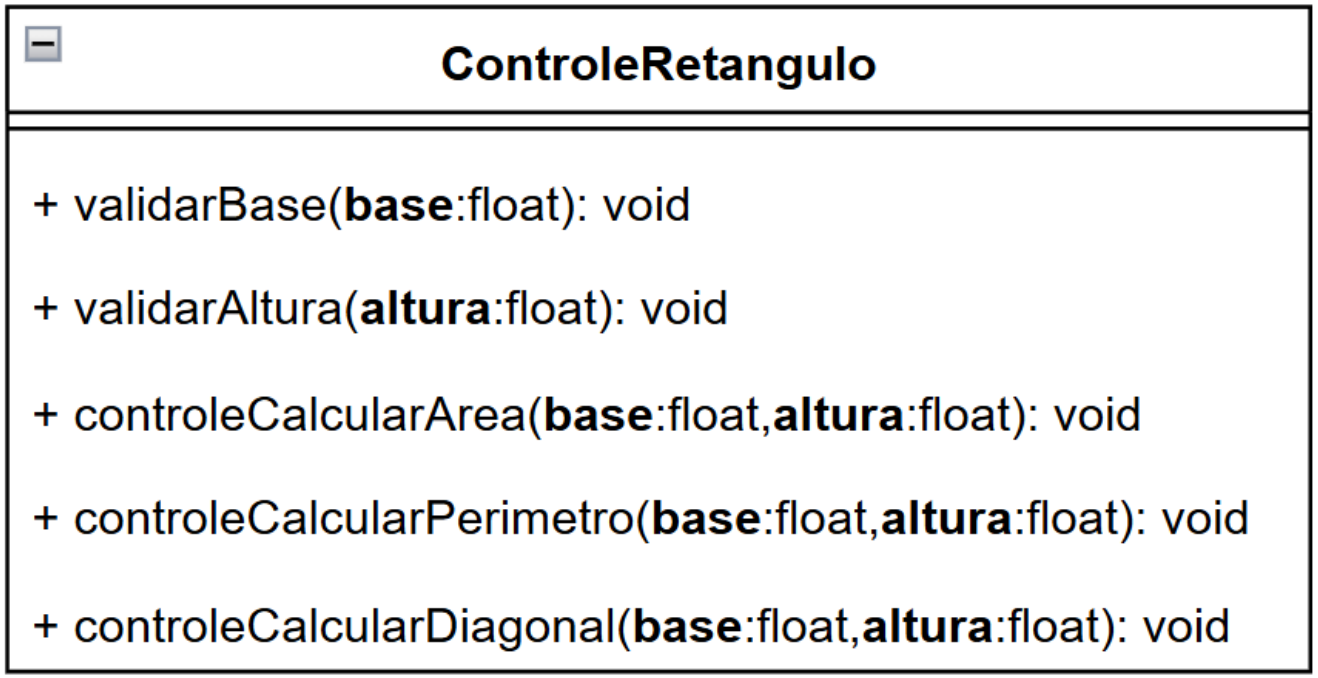
Arquivos de roteamento

- .htaccess
- Roteador.php
- Router.php



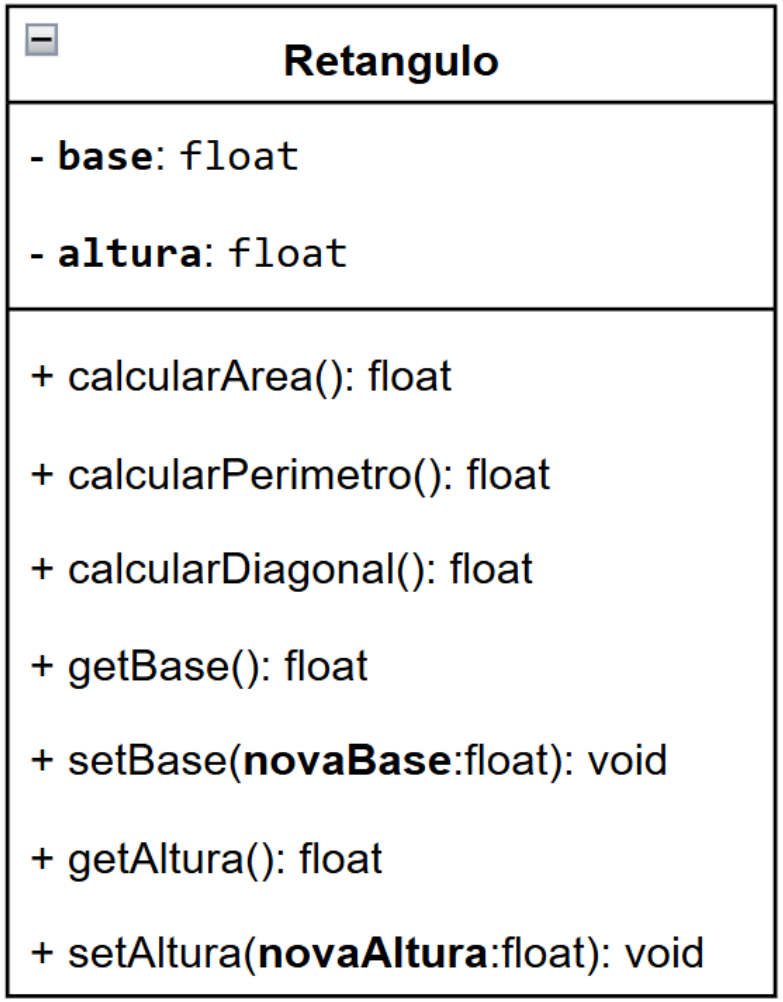
Arquivos de Controle

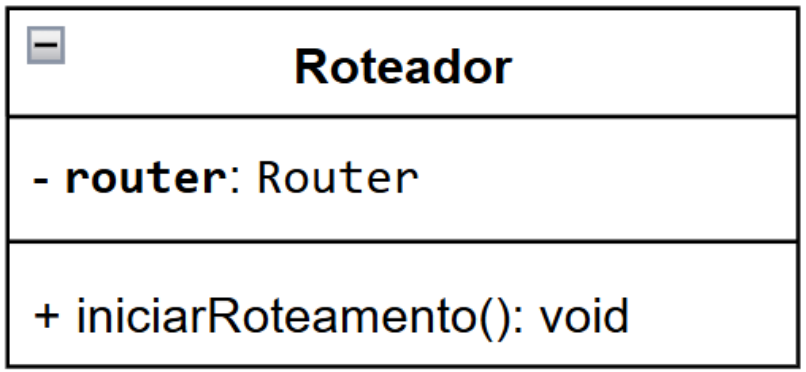
- Controle/ControleRetangulo.php
  - Observe que não há atributos na classe ControleRetangulo

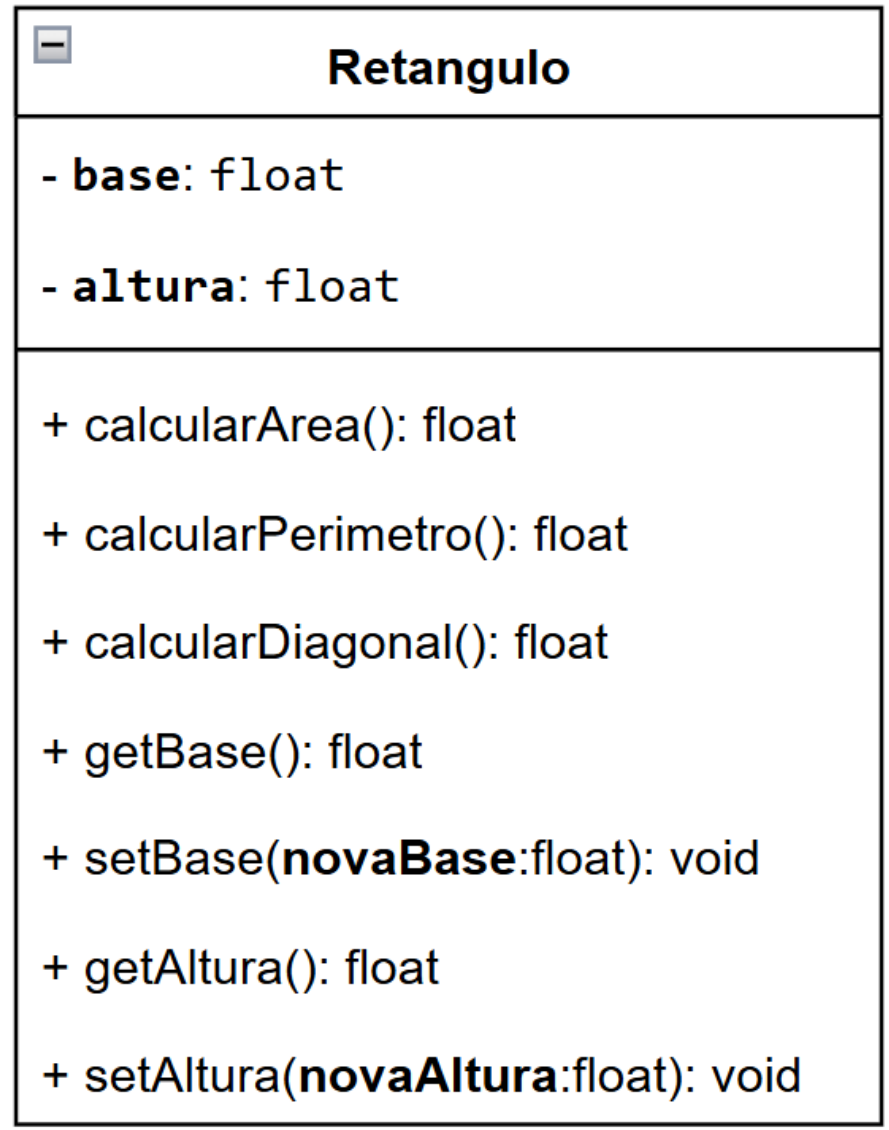


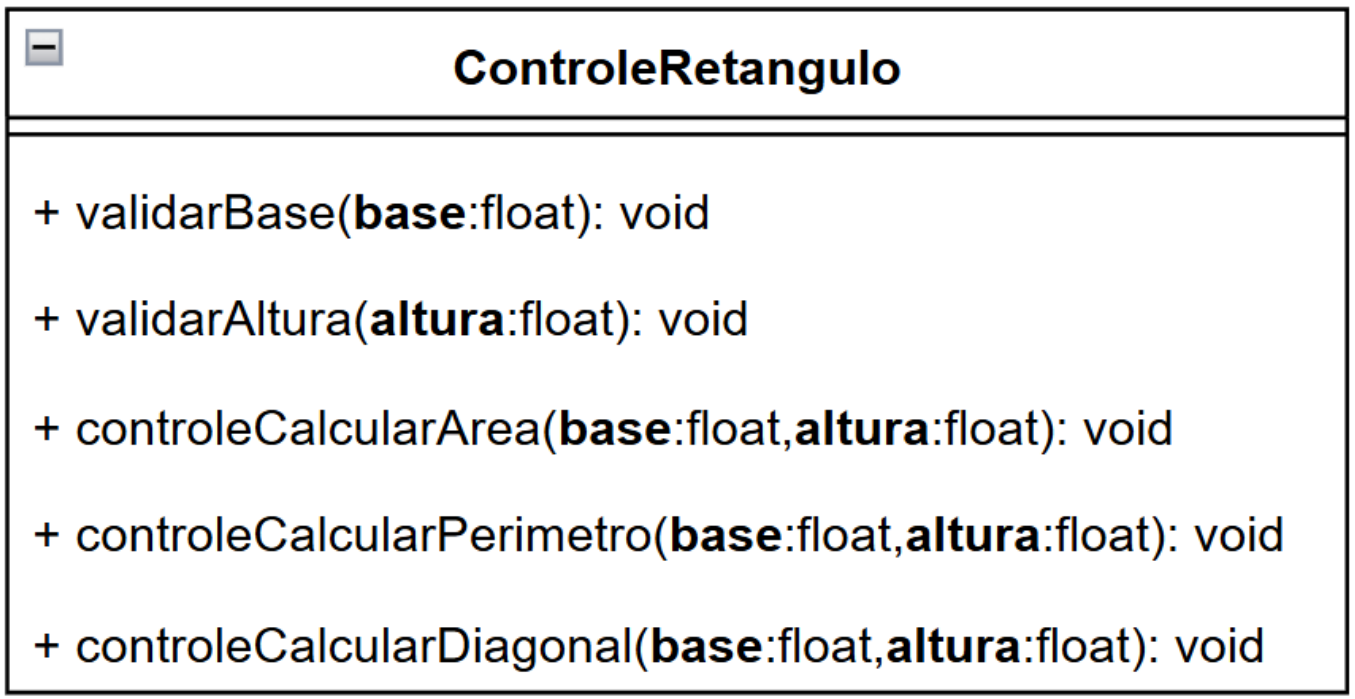
Arquivos de Modelo

- Modelo/Retangulo.php



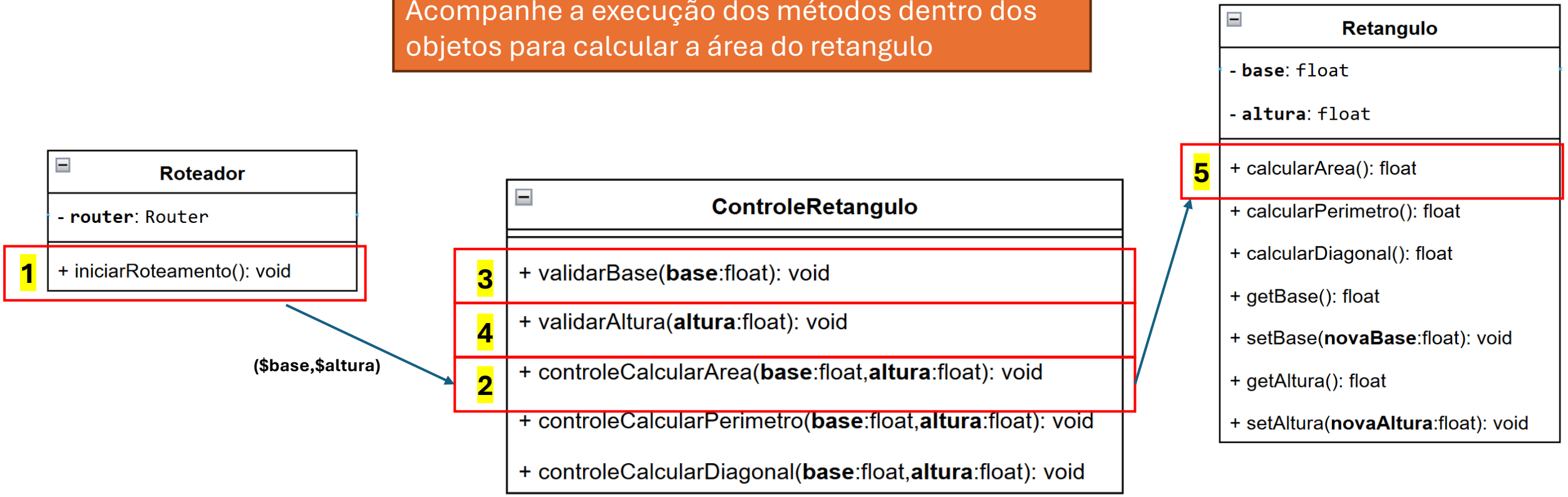

Roteador
- router: Router
+ iniciarRoteamento(): void


Retangulo
- base: float
- altura: float
+ calcularArea(): float
+ calcularPerimetro(): float
+ calcularDiagonal(): float
+ getBase(): float
+ setBase(novaBase:float): void
+ getAltura(): float
+ setAltura(novaAltura:float): void


ControleRetangulo
+ validarBase(base:float): void
+ validarAltura(altura:float): void
+ controleCalcularArea(base:float,altura:float): void
+ controleCalcularPerimetro(base:float,altura:float): void
+ controleCalcularDiagonal(base:float,altura:float): void

Fluxo de dados
http://localhost:8080/retangulos/area/10/10

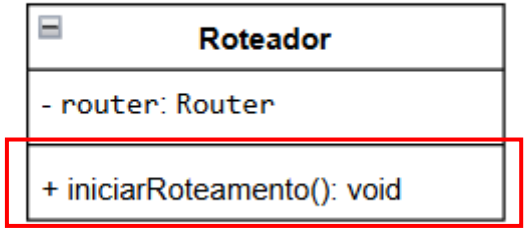
Acompanhe a execução dos métodos dentro dos objetos para calcular a área do retangulo



Roteador.php

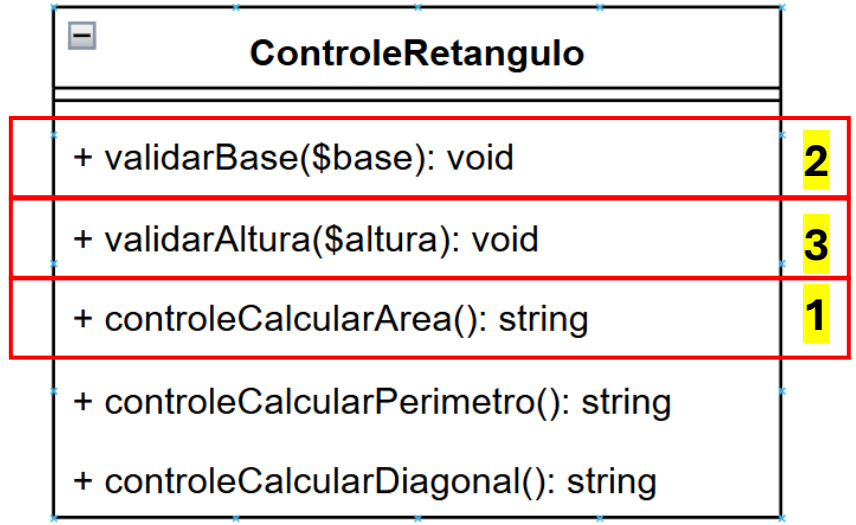
```php
<?php
require_once "Router.php"; //https://github.com/bramus/router
require_once "controle/ControleRetangulo.php";
class Roteador {
    private $router;
    public function iniciarRoteamento()    {
        $this->router = new Router(); //Cria uma instância da classe Router
        $this->router->get('/retangulos/area/(\d+)/(\d+)', function ($base, $altura) {
         header('Content-Type: application/json');
         $controle = new ControleRetangulo();
         $controle->controleCalcularArea($base, $altura);
        });

        $this->router->run(); //inicializa o roteamento
    }
}
$objetoRoteador = new Roteador();
$objetoRoteador->iniciarRoteamento();

?>
```

http://localhost:8080/retangulos/area/5/2



```php
public function controleCalcularArea($base, $altura){
    $this->validarBase($base);
    $this->validarAltura($altura);
    $objetoRetangulo = new Retangulo();
    $objetoRetangulo->setBase($base);
    $objetoRetangulo->setAltura($altura);
    $area = $objetoRetangulo->calcularArea();
    $objetoResposta = new stdClass();
    $objetoResposta->msg = "Sucesso";
    $objetoResposta->status = true;
    $objetoResposta->dados = new stdClass();
    $objetoResposta->dados->area = $area;
    $objetoResposta->dados->base = $base;
    $objetoResposta->dados->altura = $altura;
    http_response_code(200);
    $jsonResposta = json_encode($objetoResposta);
    echo  $jsonResposta;
    return  $jsonResposta;
}
```

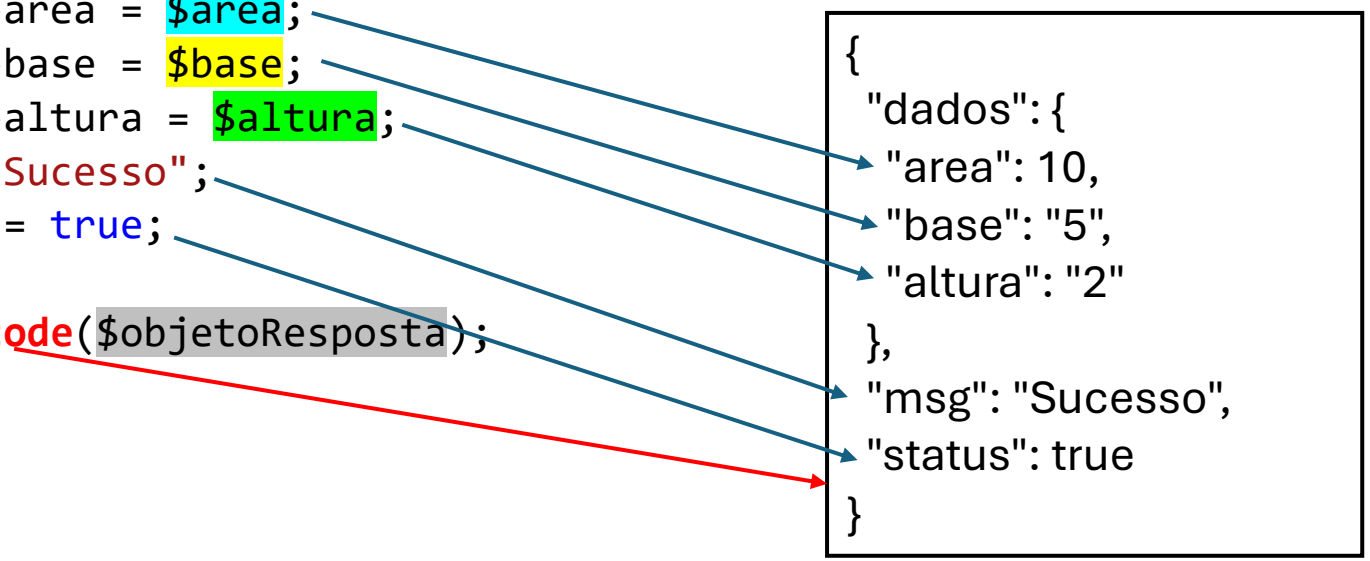
$base, $altura vem do roteador



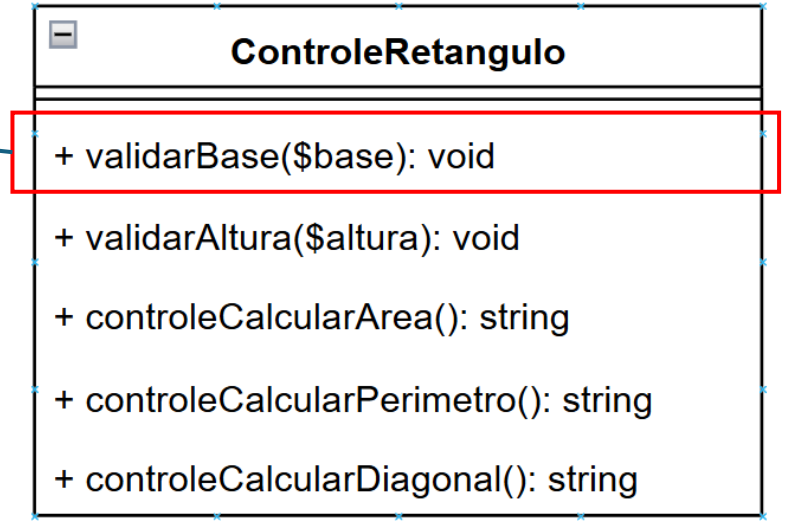
Observe que são chamados os métodos de validação e não é realizada nenhuma condicional, ou seja ao chamar
`$this->validarBase($base);`
a próxima linha só será executada
se a base for validada

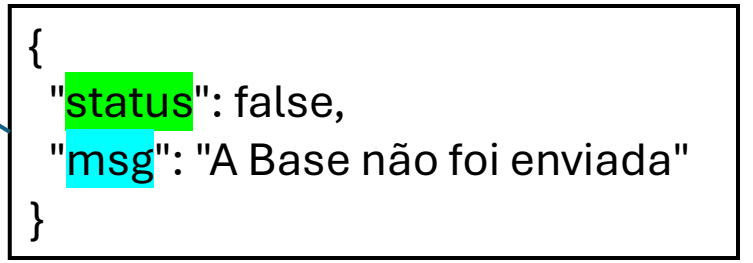
```php
public function controleCalcularArea($base, $altura)    {
    $this->validarBase($base);
    $this->validarAltura($altura);
    $objetoRetangulo = new Retangulo();
    $objetoRetangulo->setBase($base);
    $objetoRetangulo->setAltura($altura);
    $area = $objetoRetangulo->calcularArea();
    $objetoResposta = new stdClass();
    $objetoResposta->dados = new stdClass();
    $objetoResposta->dados->area = $area;
    $objetoResposta->dados->base = $base;
    $objetoResposta->dados->altura = $altura;
    $objetoResposta->msg = "Sucesso";
    $objetoResposta->status = true;
    http_response_code(200);
    $jsonResposta = json_encode($objetoResposta);
    echo  $jsonResposta;
    return  $jsonResposta;
}
```

json_encode($objetoResposta) transforma o objeto da classe stdClass() em um json



```php
public function validarBase($base)   {
    if (!isset($base)) {
        $objetoResposta = new stdClass();
        $objetoResposta->status = false;
        $objetoResposta->msg = "A Base não foi enviada";
        http_response_code(400);
        echo json_encode($objetoResposta);
        exit();
    }
    if (!is_numeric($base)) {
        $objetoResposta = new stdClass();
        $objetoResposta->status = false;
        $objetoResposta->msg = "A Base não é um número";
        http_response_code(400);
        echo json_encode($objetoResposta);
        exit();
    }
  if ($base <= 0) {
        $objetoResposta = new stdClass();
        $objetoResposta->status = false;
        $objetoResposta->msg = "A Base não pode ser menor ou igual a zero";
        http_response_code(400);
        echo json_encode($objetoResposta);
        exit();
    }
   }
```



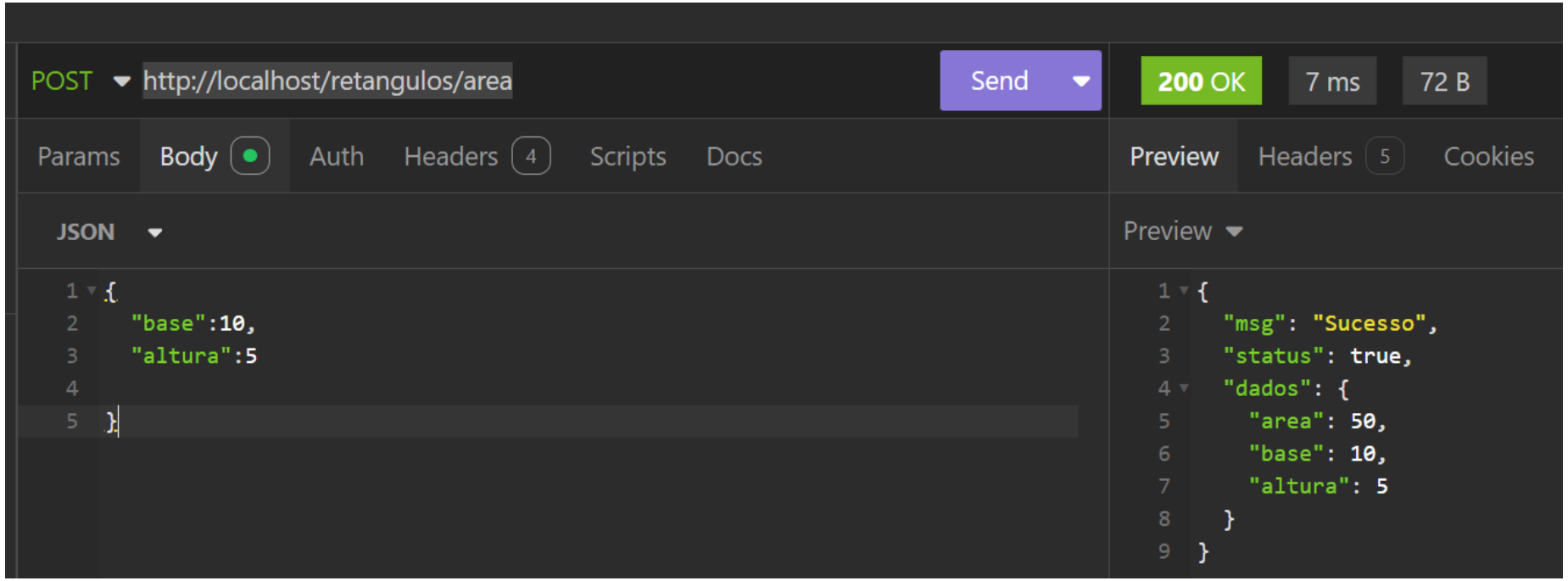


Http code: 400
Significa bad-request
Ou seja requisição ruim.
Utilizado quando são fornecidos
dados incorretos ou inválidos

Exemplo Utilizando o post

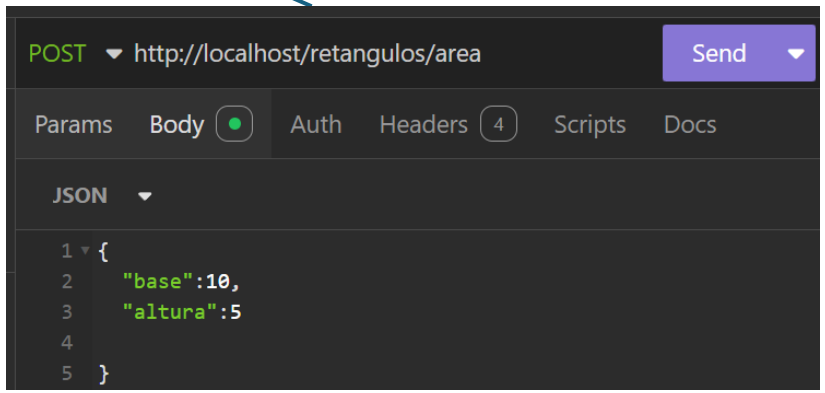
Observe que para enviar um post para um endpoint, os dados não são enviados por meio da URI, mas sim por meio do corpo da requisição no formato json

Post: http://localhost/retangulos/area



POST: /retangulos/area

```php
$this->router->post('/retangulos/area/', function () {
        //recupera o texto json enviado no corpo da requisição post
        $stringJson = file_get_contents('php://input');
        $objJson = json_decode($stringJson);      // Tenta decodificar como JSON
       // Verifica se o JSON decodificado é válido
        if (json_last_error() !== JSON_ERROR_NONE) {
            http_response_code(400); // Código 400: Bad Request
            $erro = new stdClass();
            $erro->status = false;
            $erro->erro = 'JSON inválido.';
            echo json_encode($erro);
            exit();
        }
        $base = $objJson->base;
        $altura = $objJson->altura;
        $controle = new ControleRetangulo();
        $controle->controleCalcularArea($base, $altura);
  });
```



Rotina de testes

- Antes de construir o front-end é necessário testar a API já construída.
- É uma péssima prática construir tudo e testar no final.
  - A aplicação é complexa e encontrar problemas fica quase impossível
- Apenas após funcionar no teste construa o front.