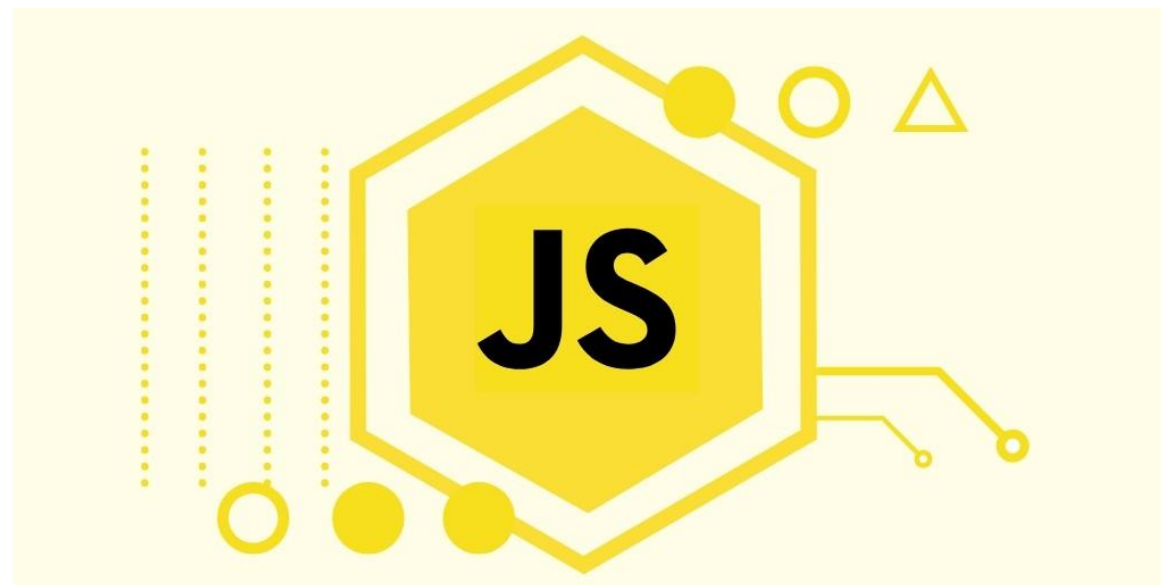


Prof. Me. Hélio Esperidião

POO – MVC e JAVASCRIPT



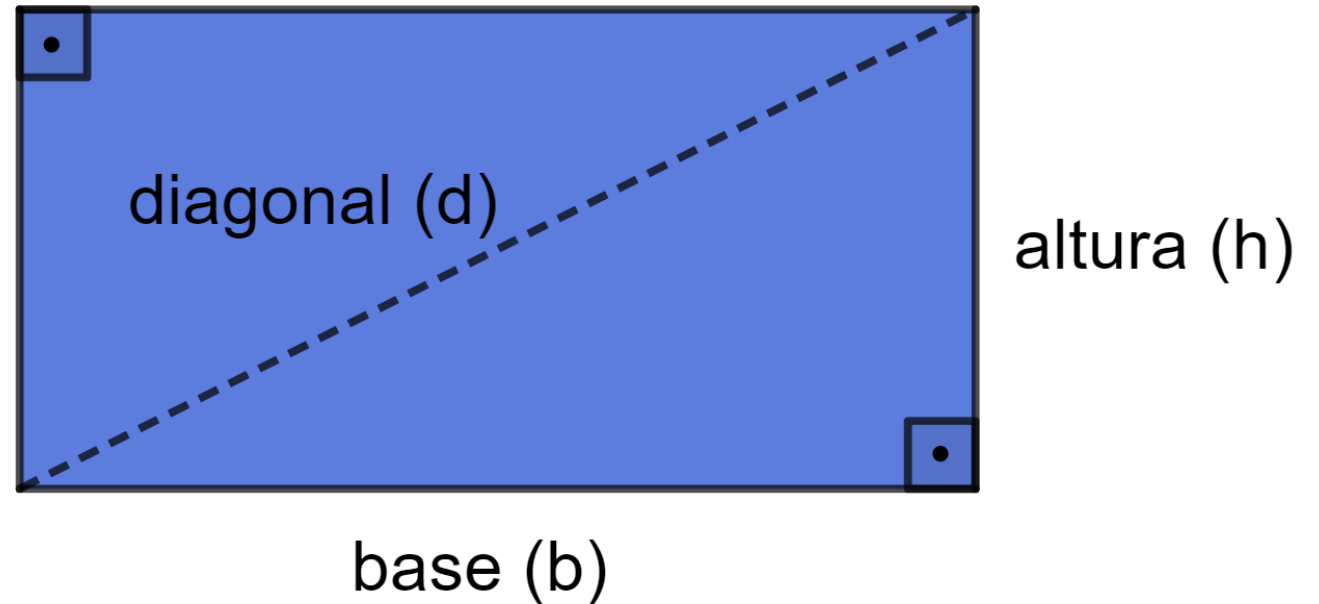
POO e Javascript no backend

O java script da
suporte a
programação
orientada a objetos.

Então é possível criar
classes e objetos

Classe (Retangulo)

- Atributos
(Características)
 - base
 - altura
- Métodos
 - CalcularArea()
 - CalcularDiagonal()
 - CalcularPerimetro()



POO

Observe que o get e o set são levemente diferentes do php
A funcionalidade é a mesma, mas a forma de escrever é diferente

```
module.exports = class Retangulo {  
  // atributos privados  
  #base;  
  #altura;  
  calcularArea() {  
    return this.#base * this.#altura;  
  }  
  calcularPerimetro() {  
    return 2 * (this.#base + this.#altura);  
  }  
  calcularDiagonal() {  
    return Math.sqrt(this.#base ** 2 + this.#altura ** 2);  
  }  
  set base(base) {  
    this.#base = base;  
  }  
  get base() {  
    return this.#base;  
  }  
  set altura(altura) {  
    this.#altura = altura;  
  }  
  get altura() {  
    return this.#altura;  
  }  
}
```

```
const ret1 = new Retangulo();  
ret1.base = 5;  
ret1.altura = 5;  
const calculo = ret1.calcularArea();  
const resposta = {  
  base: ret1.base,  
  altura: ret1.altura,  
  area: calculo  
}
```

```
module.exports = class Retangulo {  
  // atributos privados  
  #base;  
  #altura;  
  calcularArea() {  
    return this.#base * this.#altura;  
  }  
  calcularPerimetro() {  
    return 2 * (this.#base + this.#altura);  
  }  
  calcularDiagonal() {  
    return Math.sqrt(this.#base ** 2 +  
this.#altura ** 2);  
  }  
  set base(base) {  
    this.#base = base;  
  }  
  get base() {  
    return this.#base;  
  }  
  set altura(altura) {  
    this.#altura = altura;  
  }  
  get altura() {  
    return this.#altura;  
  }  
}
```

Declaração dos atributos

Get/set

```
?php  
class Retangulo{  
  private $base;  
  private $altura;  
  public function setBase($novaBase){  
    $this->base=$novaBase;  
  }  
  public function getBase(){  
    return $this->base;  
  }  
  public function setAltura($novaAltura){  
    $this->altura = $novaAltura;  
  }  
  public function getAltura(){  
    return $this->altura;  
  }  
  public function calcularArea(){  
    return ($this->altura*$this->base);  
  }  
  public function calcularDiagonal(){  
    return sqrt(pow($this->altura,2) + pow($this->base,2));  
  }  
  public function calcularPerimetro(){  
    return ($this->altura*2 + $this->base*2) ;  
  }  
}  
?>
```

Usando a classe Retangulo.js

```
const express = require('express'); // Importa o Express
const Retangulo = require('./Retangulo');
const app = express(); // Recupera uma instância do Express
const portaServico = 3000;

// Middleware para habilitar o parsing de JSON no corpo das requisições
app.use(express.json());

// Rota: GET /usuarios
app.get('/retangulos/:base/:altura', (request, response) => {
  const base = request.params.base;
  const altura = request.params.altura;
  const retangulo1 = new Retangulo();
  retangulo1.base = base;
  retangulo1.altura = altura;
  const calculo = retangulo1.calcularArea();
  const objResposta = {
    base: parseFloat(retangulo1.base),
    altura: parseFloat(retangulo1.altura),
    area: parseFloat(calculo)
  }
  response.status(200).send(objResposta);
});

// Inicia a espera por requisições HTTP
app.listen(portaServico, () => {
  console.log(`API rodando no endereço: http://localhost:${portaServico}/`);
});
```

```
app.get('/retangulos/:base/:altura', (request, response) => {  
});
```

The screenshot displays a web browser's developer tools interface. The top section shows a GET request to `http://localhost:3000/retangulos/10/20`. The status is `200 OK`, with a response time of `41 ms` and a size of `34 B`. The `Body` tab is selected, showing the response in `JSON` format. The response body is a JSON object: `{ "base": 10, "altura": 20, "area": 200 }`. Arrows from the code above point to the corresponding parts of the URL in the browser's address bar.

Method	URL	Status	Time	Size
GET	http://localhost:3000/retangulos/10/20	200 OK	41 ms	34 B

Params | **Body** | Auth | Headers (4) | Scripts | Docs

JSON

```
1 ...  
2 {  
3   "base": 10,  
4   "altura": 20,  
5   "area": 200  
6 }
```

MVC

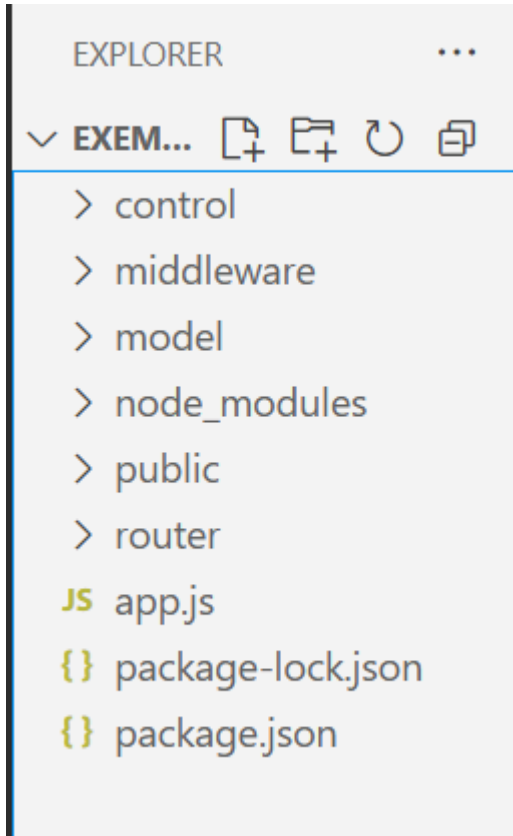
- O padrão MVC (Model-View-Controller) é uma arquitetura de software que separa a aplicação em três componentes principais:
 - Model (Modelo): Classes
 - Responsável pela lógica de negócios e pela interação com o banco de dados.
 - Contém os dados da aplicação e as regras de negócio.
 - Notifica a View quando há mudanças nos dados.
 - View (Visão): frontend
 - Responsável pela interface do usuário e pela exibição dos dados.
 - Representa os dados do Model de forma visual.
 - Recebe a interação do usuário e repassa ao Controller.
 - Controller (Controlador): roteamento e utilização das classes
 - Atua como intermediário entre o Model e a View.
 - Processa as entradas do usuário vindas da View e atualiza o Model.
 - Pode também atualizar a View com base nas mudanças no Model.

Observe: onde está o controle e onde está o roteamento?

```
// Rota: GET /usuarios
app.get('/retangulos/:base/:altura', (request, response) =>
{
  const base = request.params.base;
  const altura = request.params.altura;
  const retangulo1 = new Retangulo();
  retangulo1.base = base;
  retangulo1.altura = altura;
  const calculo = retangulo1.calcularArea();
  const objResposta = {
    base: parseFloat(retangulo1.base),
    altura: parseFloat(retangulo1.altura),
    area: parseFloat(calculo)
  }
  response.status(200).send(objResposta);
});
```

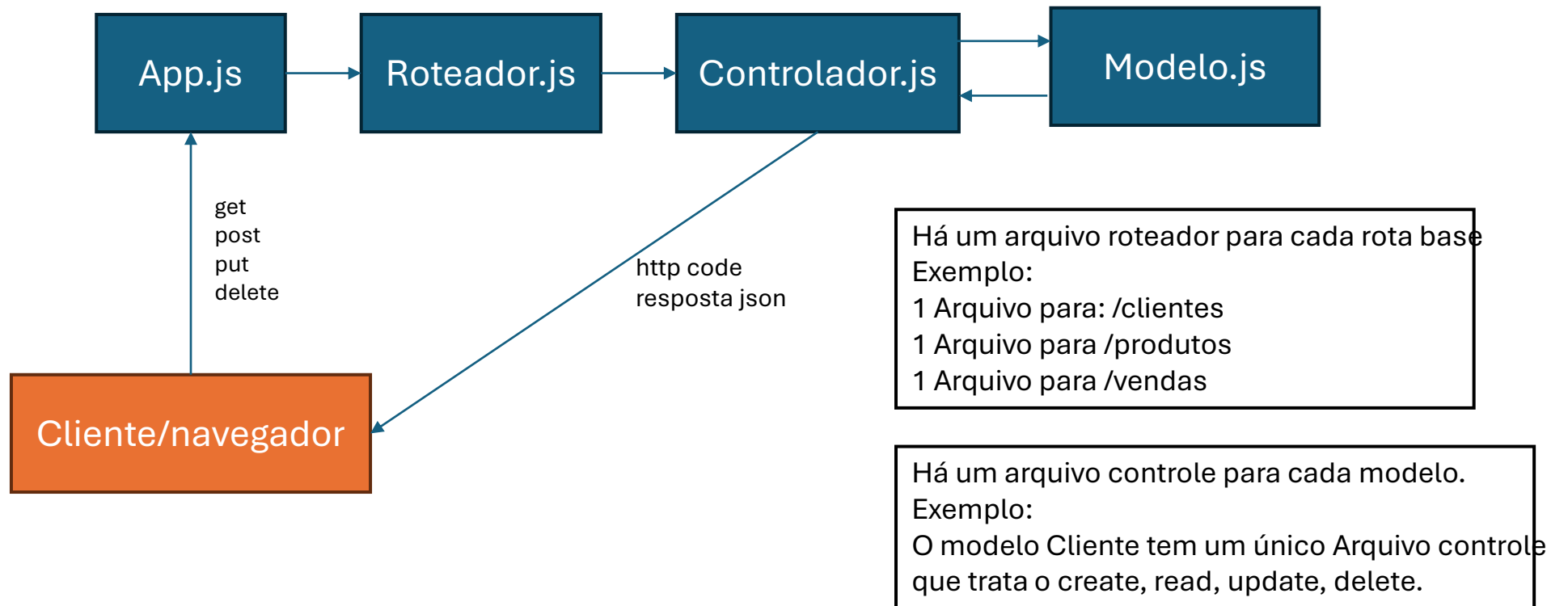
- O roteamento e controle estão no mesmo arquivo, é comum em uma aplicação Javascript separar o roteamento do controle.

Arquitetura de pastas

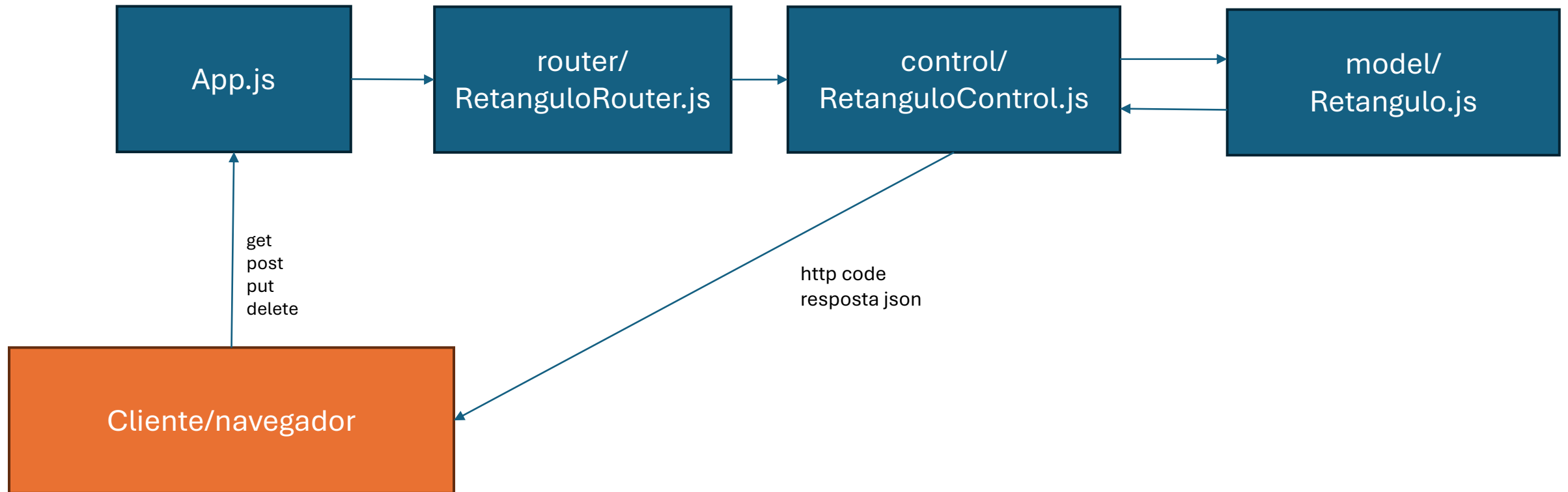


- Construa uma pasta para cada funcionalidade do sistema
 - Model
 - Public => camada view
 - Control
 - Router //para o roteamento

Arquitetura básica:



Arquitetura básica na prática:



App.js

```
const express = require('express'); // Importa o framework Express
// Importa a classe de roteamento do Retângulo
const RouterRetangulo = require('./router/RetanguloRouter');
// Cria uma instância do Express, que servirá como servidor HTTP
const app = express();
// Porta em que o serviço ficará disponível
const portaServico = 3000;
app.use(express.json()); // Middleware responsável por habilitar o parsing de JSON

// Define a pasta "public" como diretório de arquivos estáticos
// Todos os arquivos dentro de "public" poderão ser acessados diretamente via URL
app.use(express.static('public'));

const roteadorRetangulo = new RouterRetangulo(); //tratar todas as requisições para `/retangulos`

app.use(
  '/retangulos',
  roteadorRetangulo.getRouter() // Método que retorna o Router configurado para tratar as requisições
);
app.listen(portaServico, () => {
  console.log(`API rodando no endereço: http://localhost:${portaServico}/`);
});
```

RetanguloRouter.js

```
const express = require('express');
const RetanguloControl = require('../control/RetanguloControl');
//Classe responsável por configurar as rotas de /retangulos.
module.exports = class RetanguloRouter {
  #router; // atributo privado
  #retanguloControl; // atributo privado
  getRouter() {
    this.#router = express.Router();
    this.#retanguloControl = new RetanguloControl();
    // rota GET: /retangulos/:base/:altura
    this.#router.get('/:base/:altura',
      this.#retanguloControl.calcularTudo
    );
    this.#router.get('/areas/:base/:altura',
      this.#retanguloControl.calcularArea
    );
    // rota POST (ainda não implementada)
    this.#router.post(
     ('/:base/:altura',
        (request, response) => response.status(501).send({ message: "Rota POST ainda não implementada." })
      );
    return this.#router;
  }
}
```

```
const Retangulo = require('../model/Retangulo');

module.exports = class ControlRetangulo {
  calcularTudo(request, response) {
    const base = request.params.base;
    const altura = request.params.altura;
    const retangulo1 = new Retangulo();
    retangulo1.base = base;
    retangulo1.altura = altura;
    const objResposta = {
      retangulo: {
        perimetro: retangulo1.calcularPerimetro(),
        diagonal: retangulo1.calcularDiagonal(),
        area: retangulo1.calcularArea()
      }
    };
    response.status(200).send(objResposta);
  }
  calcularArea(request, response) {
    const base = request.params.base;
    const altura = request.params.altura;
    const retangulo1 = new Retangulo();
    retangulo1.base = base;
    retangulo1.altura = altura;

    response.status(200).send({
      retangulo: {
        base: parseInt(retangulo1.base),
        altura: parseInt(retangulo1.altura),
        area: parseInt(retangulo1.calcularArea())
      }
    });
  }
}
```

ControlRetangulo.js

```
module.exports = class Retangulo {  
  // atributos privados  
  #base;  
  #altura;  
  calcularArea() {  
    return this.#base * this.#altura;  
  }  
  calcularPerimetro() {  
    return 2 * (this.#base + this.#altura);  
  }  
  calcularDiagonal() {  
    return Math.sqrt(this.#base ** 2 + this.#altura ** 2);  
  }  
  set base(base) {  
    this.#base = base;  
  }  
  get base() {  
    return this.#base;  
  }  
  set altura(altura) {  
    this.#altura = altura;  
  }  
  get altura() {  
    return this.#altura;  
  }  
}
```

Retangulo.js

O Que é Middleware?

- Middleware é um conceito fundamental no desenvolvimento de aplicações web, especialmente em frameworks como Express.js no Node.js.
- Um middleware é essencialmente uma função que se coloca no caminho do ciclo de processamento de uma solicitação HTTP, com o propósito de interagir com a requisição, a resposta ou ambos.

Middleware

- Middleware é uma função que tem acesso aos objetos de requisição (request), resposta (response), e a uma função next na aplicação Express.
- Pode executar qualquer código, modificar os objetos de requisição e resposta, encerrar o ciclo de requisição/resposta, ou chamar a próxima função de middleware na pilha.

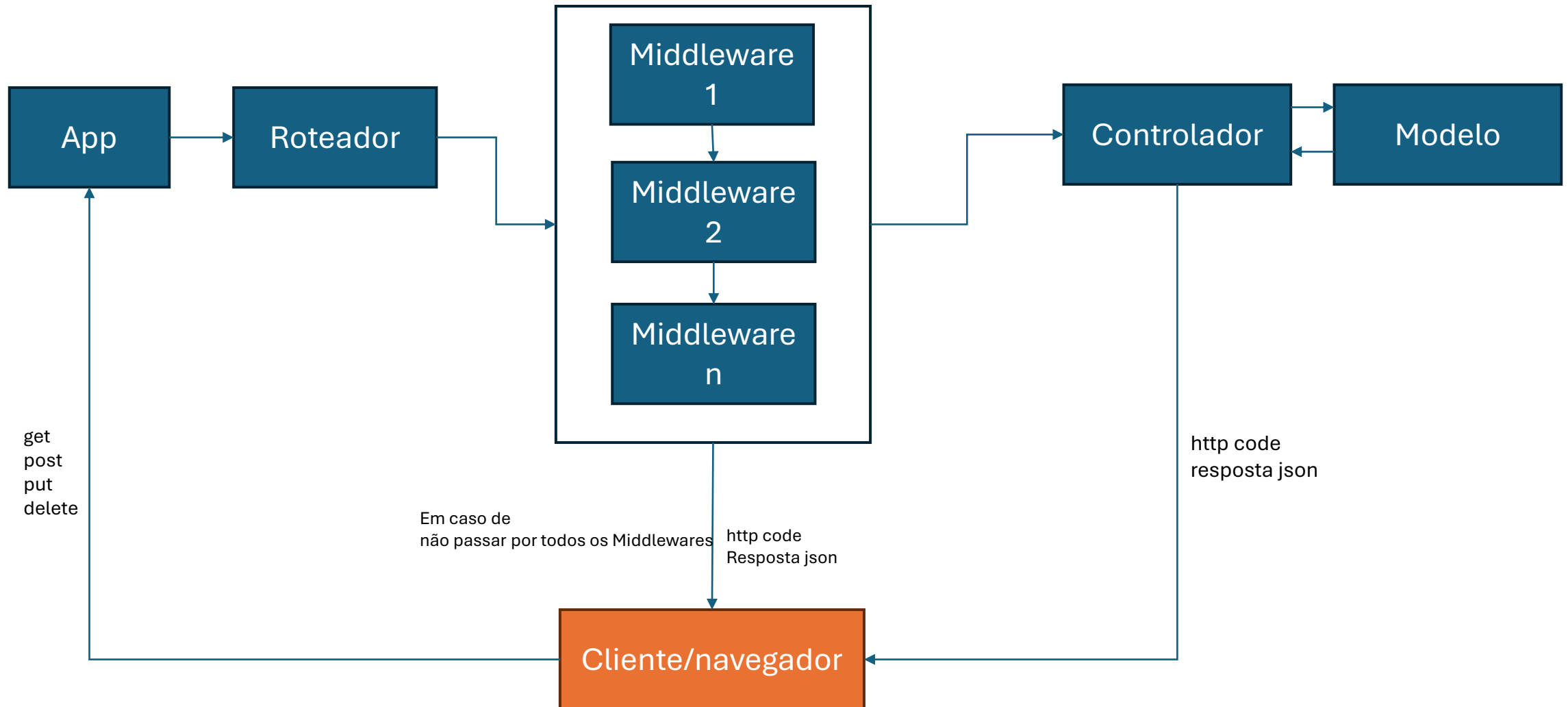
Para Que Serve o Middleware?

- Middleware pode ser usado para processar os dados da solicitação, como o corpo da requisição, cookies, cabeçalhos, etc. Por exemplo, `express.json()` e `express.urlencoded()` são middlewares que analisam o corpo da requisição para JSON e dados codificados em URL, respectivamente.
- **Tipicamente pode ser utilizado para validar dados e regras de negócio antes da chamada de um controle.**

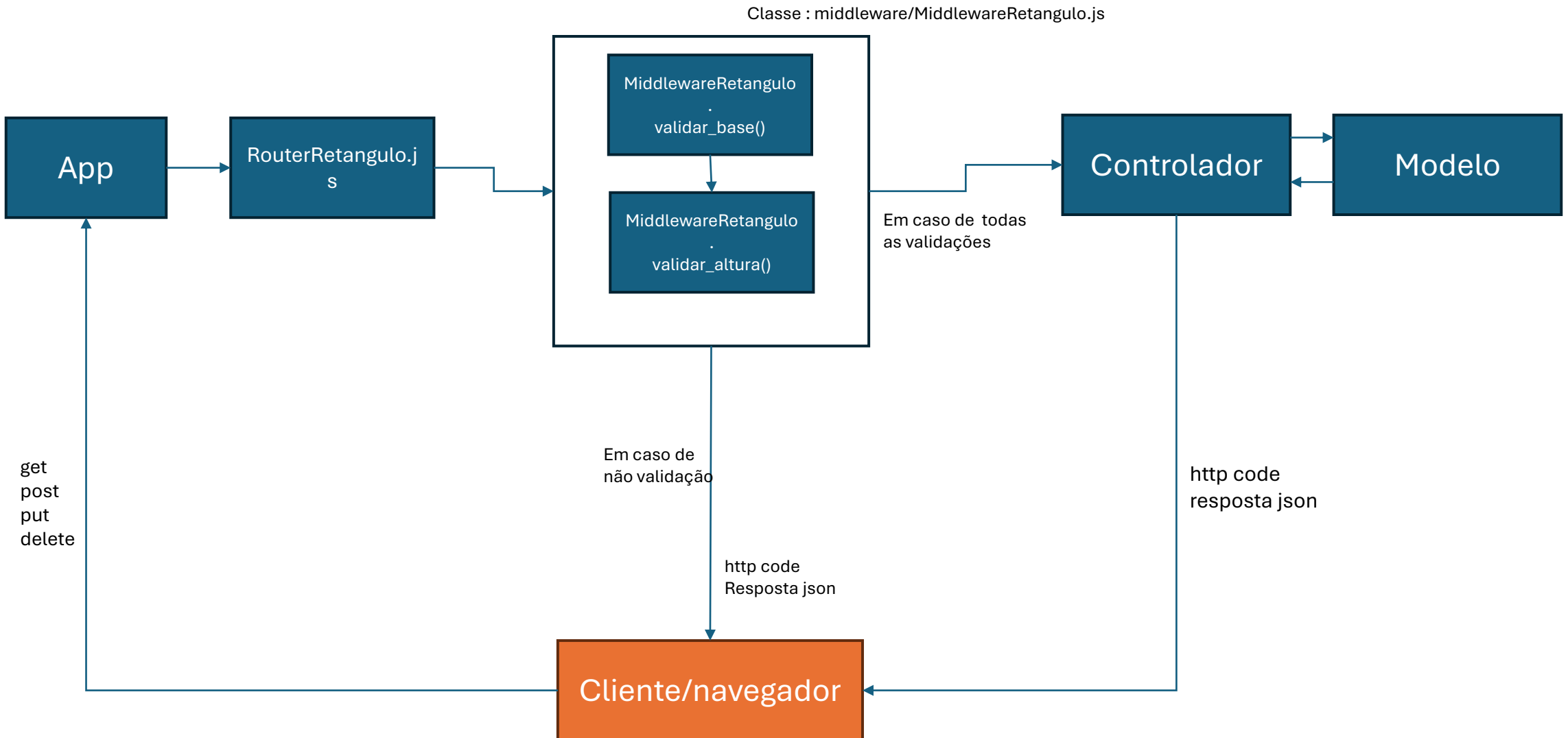
Qual seria o Middleware para o exemplo do Retângulo?

- A ideia é segmentar o processamento em múltiplas partes.
 - 1 Middleware para validar a base
 - 1 Middleware para validar a altura.
- Os dois Middlewares validam as entradas de dados para somente depois acessar um controle específico.

Arquitetura básica:



Arquitetura Retângulo:



```
const express = require('express');
const RetanguloControl = require('../control/RetanguloControl');
const RetanguloMiddleware = require('../middleware/RetanguloMiddleware');
module.exports = class RetanguloRouter {
  #router;
  #retanguloControl;
  #retanguloMiddleware;
  getRouter() {
    this.#router = express.Router();
    this.#retanguloControl = new RetanguloControl();
    this.#retanguloMiddleware = new RetanguloMiddleware();
    this.#router.get('/:base/:altura',
      this.#retanguloMiddleware.validateBaseParam,
      this.#retanguloMiddleware.validateAlturaParam,
      this.#retanguloControl.calcularTudo
    );
    this.#router.get('/:areas/:base/:altura',
      this.#retanguloMiddleware.validateBaseParam,
      this.#retanguloMiddleware.validateAlturaParam,
      this.#retanguloControl.calcularArea
    );
    this.#router.post('/:base/:altura',
      (request, response) => response.status(501).send({ message: "Rota POST ainda não implementada." })
    );
    return this.#router;
  }
}
```

```
module.exports = class RetanguloMiddleware {  
  validateBaseParam(request, response, next) {  
    const base = request.params.base;  
    if (isNaN(base)) {  
      const objResposta = {  
        success: false,  
        msg: "A base deve ser um número"  
      };  
      response.status(400).send(objResposta);  
    } else {  
      next();  
    }  
  }  
  validateAlturaParam(request, response, next) {  
    const altura = request.params.altura;  
    if (isNaN(altura)) {  
      const objResposta = {  
        success: false,  
        msg: "A altura deve ser um número"  
      };  
      response.status(400).send(objResposta);  
    } else {  
      next();  
    }  
  }  
}
```