# INTRODUÇÃO AO JAVASCRIPT NO BACKEND E NODE.JS

Prof. Me. Hélio Esperidião

# O que é Node.js?

**Node.js não é uma linguagem de programação**

Você programa utilizando a linguagem JavaScript.

Possui semelhanças com linguagens compiladas, uma vez que uma máquina virtual faz etapas de pré-compilação e otimização antes do código entrar em operação.

# O que é Node.js?

Node.js não é um framework Javascript

É uma aplicação na qual você escreve seus programas com Javascript, estes serão compilados, otimizados e interpretados pela **máquina virtual V8**.

A VM é a mesma que o Google utiliza para executar Javascript no Chrome, e foi a partir dela que surgiu do Node.js

# Para que serve Node.js?

Pode substituir java, c#, php, etc no back end de aplicações.

Node.js serve para fazer APIs.

Esse talvez seja o principal uso da tecnologia, uma vez que por default ela apenas sabe processar requisições.

Não apenas por essa limitação, mas também porque seu modelo não bloqueante de tratar as requisições o torna excelente para essa tarefa consumindo pouquíssimo hardware.

# Para que serve Node.js?

Node.js serve para fazer backend de jogos, IoT e apps de mensagens.

Node.js para APIs nestas circunstâncias (backend-as-a-service) devido ao alto volume de requisições que esse tipo de aplicações efetuam.

# Suporte a banco de dados

Bancos de Dados: você pode utilizar tanto com bancos relacionais quanto com não-relacionais:

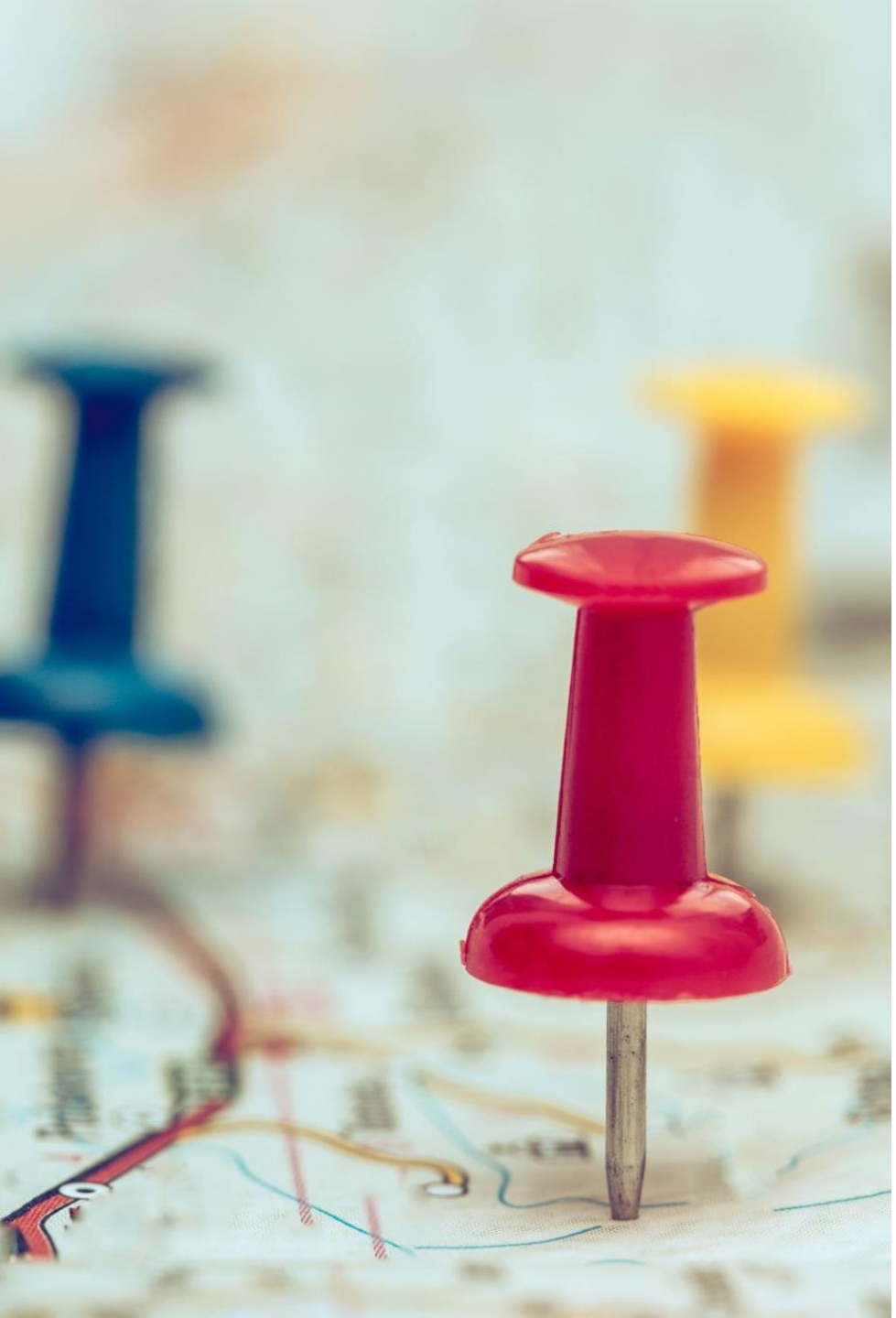
- MySQL
- PostgreSQL
- MS SQL Server
- MongoDB
- Redis
- SQLite

# Quem usa Node.js?

- Netflix
- Linkedin
- Walmart
- Trello
- Uber
- PayPal
- eBay
- NASA

# Quais as desvantagens do Javascript no backend?

- Não implementa orientação objeto a objeto com todos os recursos de linguagens como java, c# ou php.



# Download

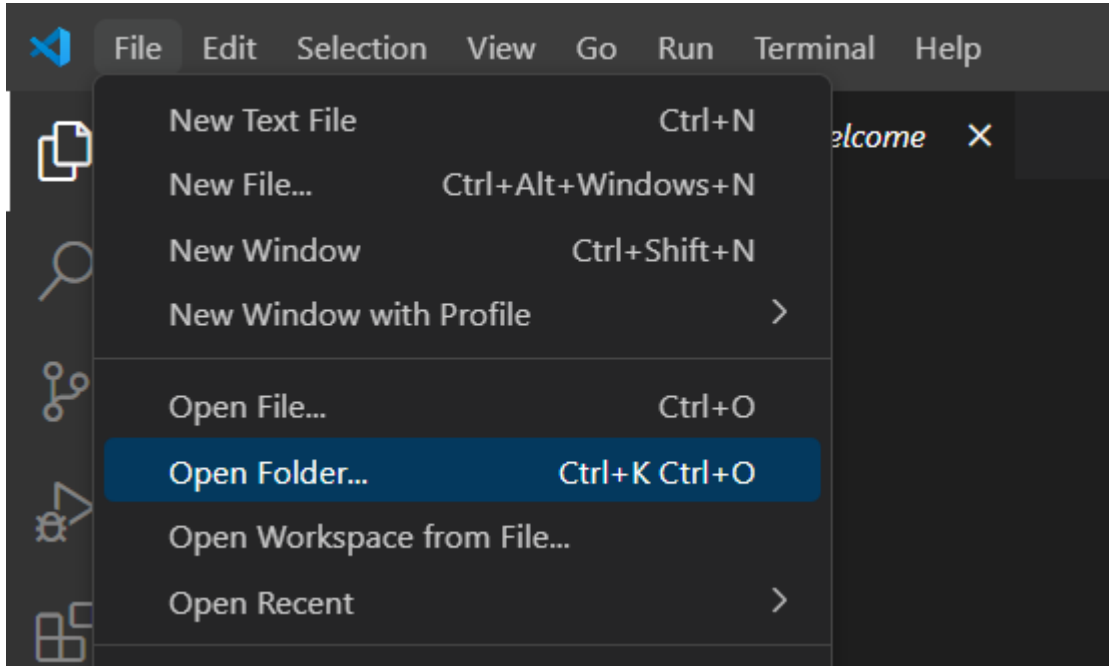
- https://nodejs.org/en/download/
  - Escolha uma versão LTS
  - LTS significa "Long Term Support" (Suporte de Longo Prazo).
    - É um termo comum no mundo do software, especialmente em sistemas operacionais, frameworks, e bibliotecas. Versões LTS de um software são mantidas por um período mais longo, recebendo atualizações de segurança e correções de bugs, mas geralmente sem a inclusão de novos recursos. Essas versões são preferidas em ambientes de produção, onde a estabilidade e a segurança são prioritárias.

# Express

- O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicações web.

# Exemplo na prática

# Faça a abertura de qualquer pasta no visual studio code



# Abra o terminal e digite: npm init

- Configura uma aplicação para rodar no node.
  - Preencha o “formulário” com as informações requisitadas

```
Press ^C at any time to quit.
package name: (exemplo01) exempo01
version: (1.0.0) 1.0.0
description: Primeiras estapas
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Hélio Esperidião
license: (ISC)
About to write to D:\professor\2024\cti\paw\4 Bimestre\exemplosJS\Exemplo01\package.json:

{
  "name": "exempo01",
  "version": "1.0.0",
  "description": "Primeiras estapas",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Hélio Esperidião",
  "license": "ISC"
}


Is this OK? (yes) yes
```

# package.json

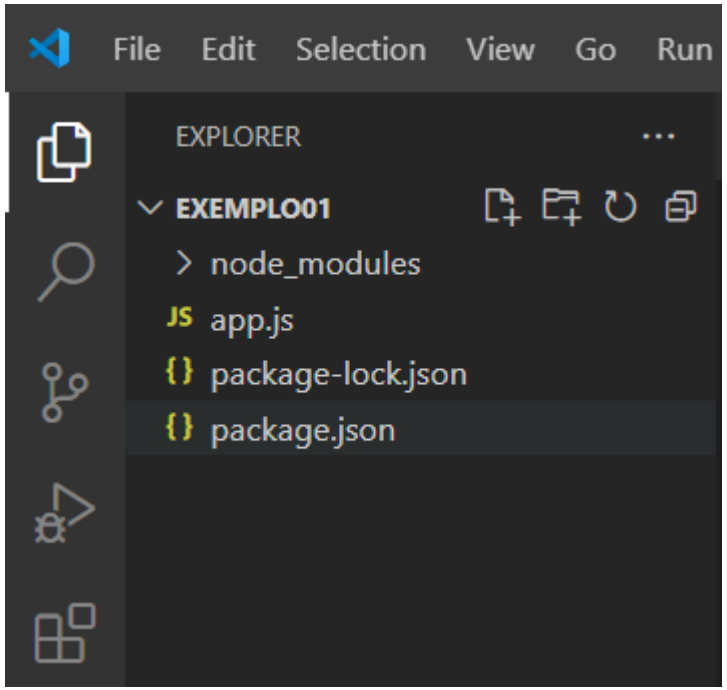
- O arquivo package.json é um arquivo fundamental em projetos Node.js.
- Ele serve como uma espécie de "manual" do seu projeto, contendo metadados sobre o projeto, como o nome, versão, autor, licença, etc

# Framework: Express

- O Express é um framework web minimalista e flexível para Node.js, projetado para criar aplicativos web e APIs de forma rápida e eficiente.
- Ele simplifica o desenvolvimento de servidores web ao fornecer um conjunto de recursos robustos e prontos para uso, sem adicionar complexidade desnecessária.
  - Rotas (Routing):
  - Middleware
  - Templates
  - Facilidade na Criação de APIs
  - Suporte a Múltiplos Protocolos

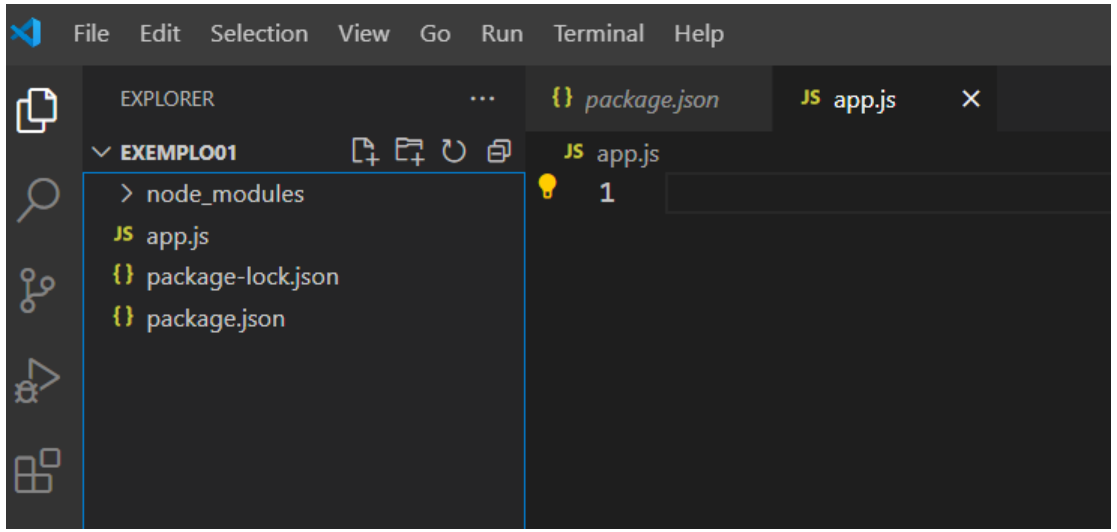
# Para instalar o express

- Digite no terminal : **npm install express --save**
- Após a instalação de qualquer pacote é criada uma pasta chamada node_modules, dentro dessa pasta há tudo o que é necessário para a sua aplicação rodar.
- Como você instalou o pacote de express, dentro da pasta há todos os arquivos do express.
  - Você não precisa salvar a pasta node_modules para “transportar” o seu projeto, pois é uma pasta que ocupa muito em disco.
  - Tipicamente os programadores preferem não salvar a pasta e usar o comando npm install, que baixa e instala todos os pacotes novamente.



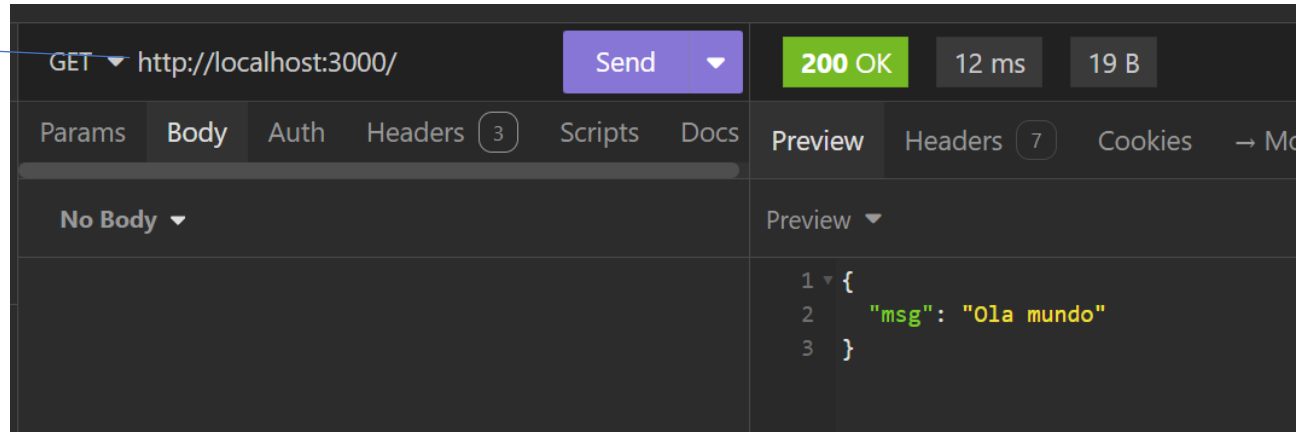
# Crie o arquivo app.js

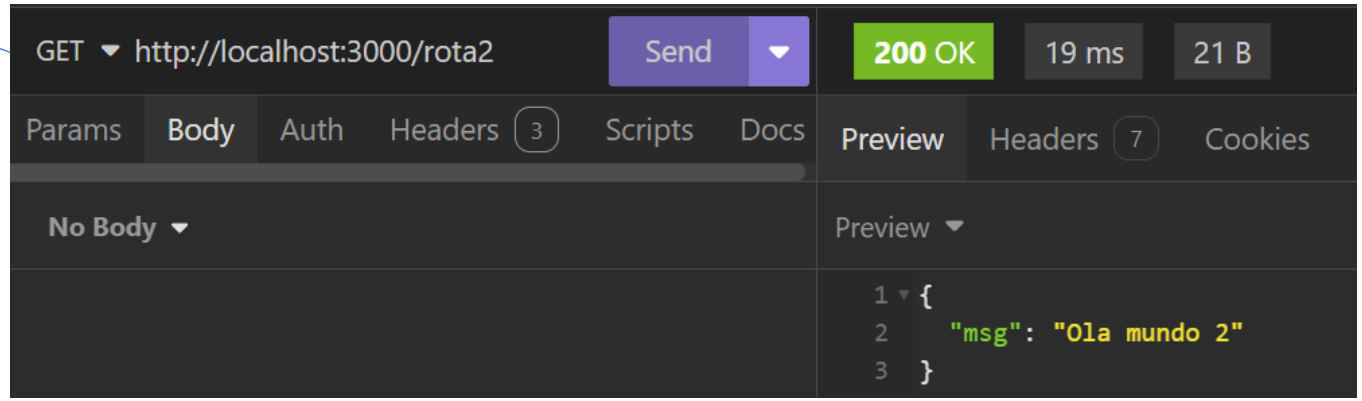
- Esse arquivo será a porta de entrada para a sua aplicação é por meio dele que serão acessados e inicializados os recursos da sua aplicação.
- O nome do arquivo em si é pouco relevante, programadores tipicamente usam index.js, app.js, main.js



# app.js

```
const express = require('express'); //importa o express
const app = express();              //recupera uma instancia de express
const portaServico = 3000;
//rota: GET: /
app.get('/', (request, response) => {
    const resposta = { msg: 'Ola mundo' }
    response.status(200).send(resposta);

});

//rota: GET: /rota2
app.get('/rota2', (request, response) => {
    const resposta = { msg: 'Ola mundo 2' }
    response.status(200).send(resposta);
});

//inicia a espera por requisições http na porta 3000
app.listen(portaServico);
console.log("Api rodando no endereço: http:localhost:3000/")
```



# Rodando a aplicação

- Para rodar a aplicação digite no console:
  - node app.js
- Toda vez que modificar o código é necessário parar o programa atual e reiniciar a aplicação.
  - Para parar a aplicação digite no console:
    - control+c
    - digite novamente: node app.js

# Pouco produtivo

- Parar a aplicação e rodar novamente toda vez que ocorrer uma modificação no código é extremamente improdutivo.

# nodemon


- O nodemon é uma biblioteca que ajuda no desenvolvimento de sistemas com o Node.js reiniciando automaticamente o servido
- Instale:
  - npm install --save-dev nodemon
- Inicie a aplicação utilizando no nodemoon:
  - npx nodemon app.js
- Sempre que houver uma modificação no código automaticamente o servidor será reiniciado.

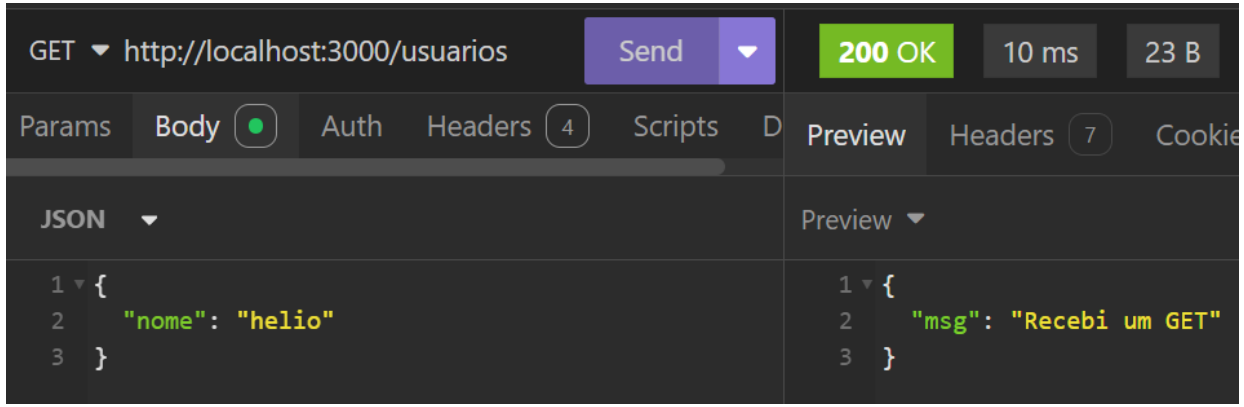
# Entendendo o: request, response

```
app.get('/', (request, response) => {
  res.send('Ola mundo');
});
```

- request é um objeto que possui dados da requisição
- response é um objeto que possui dados referente a respota.
- get: verbo http pode ser substituído por: post, put, delete, etc.

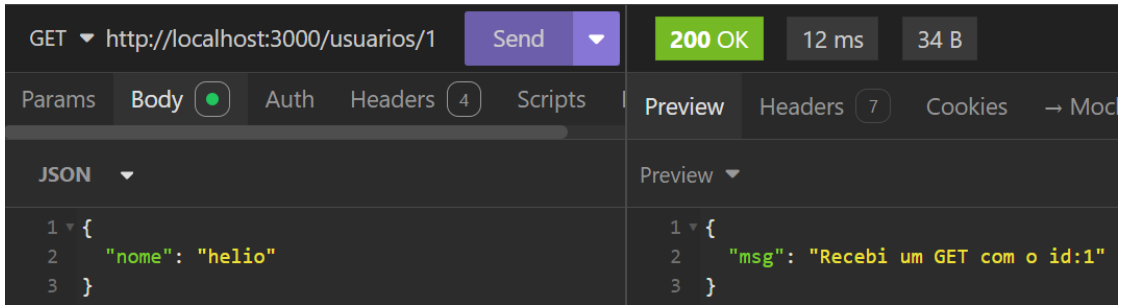
# Roteamento completo: 1/6

```
const express = require('express'); // Importa o Express
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('/usuarios', (request, response) => {
    const resposta = { msg: 'Recebi um GET' };
    response.status(200).send(resposta);
});
```



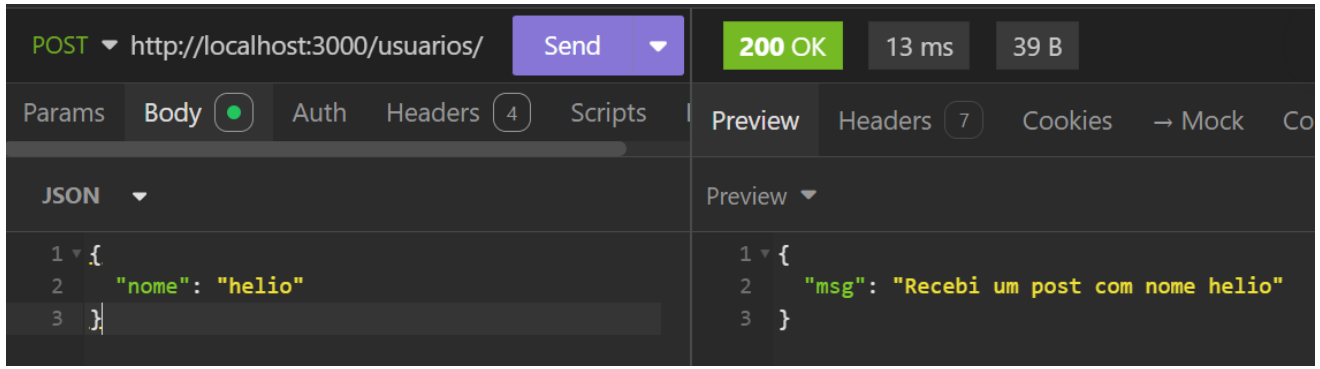
# Roteamento completo: 2/6

```
// Rota: GET /usuarios/id
app.get('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const resposta = { msg: 'Recebi um GET com o id:' + id };
    response.status(200).send(resposta);
});
```



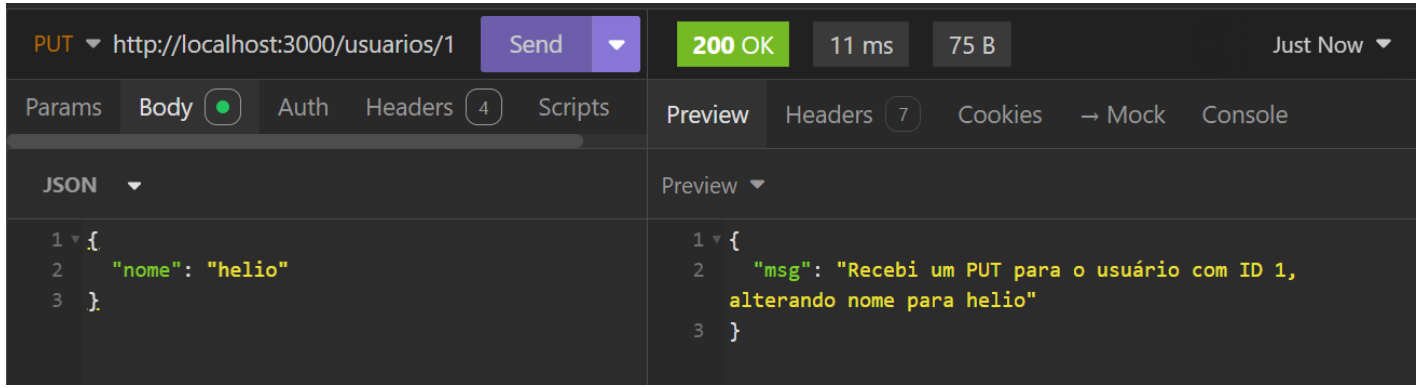
# Roteamento completo: 3/6

```
// Rota: POST /usuarios
app.post('/usuarios', (request, response) => {
    const nome = request.body.nome; // Exemplo de dado enviado no corpo da
    const resposta = { msg: `Recebi um post com nome ${nome}` };
    response.status(200).send(resposta);
});
```



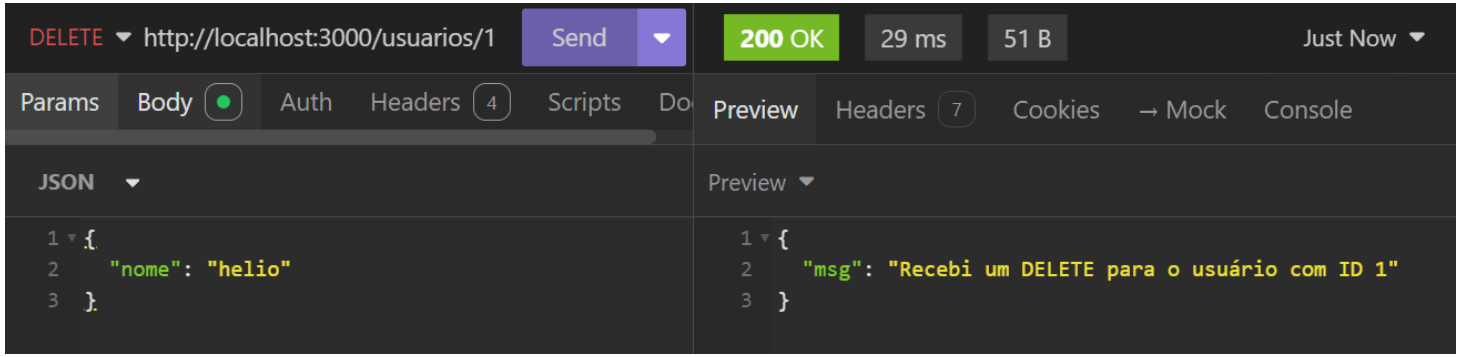
# Roteamento completo: 4/6

```
// Rota: PUT /usuarios/:id
app.put('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const nome = request.body.nome; // Exemplo de dado enviado no corpo da requisição
    const resposta = { msg: `Recebi um PUT para o usuário com ID ${id}, alterando nome para ${nome}` };
    response.status(200).send(resposta);
});
```



# Roteamento completo: 5/6

```
// Rota: DELETE /usuarios/:id
app.delete('/usuarios/:id', (request, response) => {
    const id = request.params.id;
    const resposta = { msg: `Recebi um DELETE para o usuário com ID ${id}` };
    response.status(200).send(resposta);
});
```



# Roteamento completo: 6/6

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

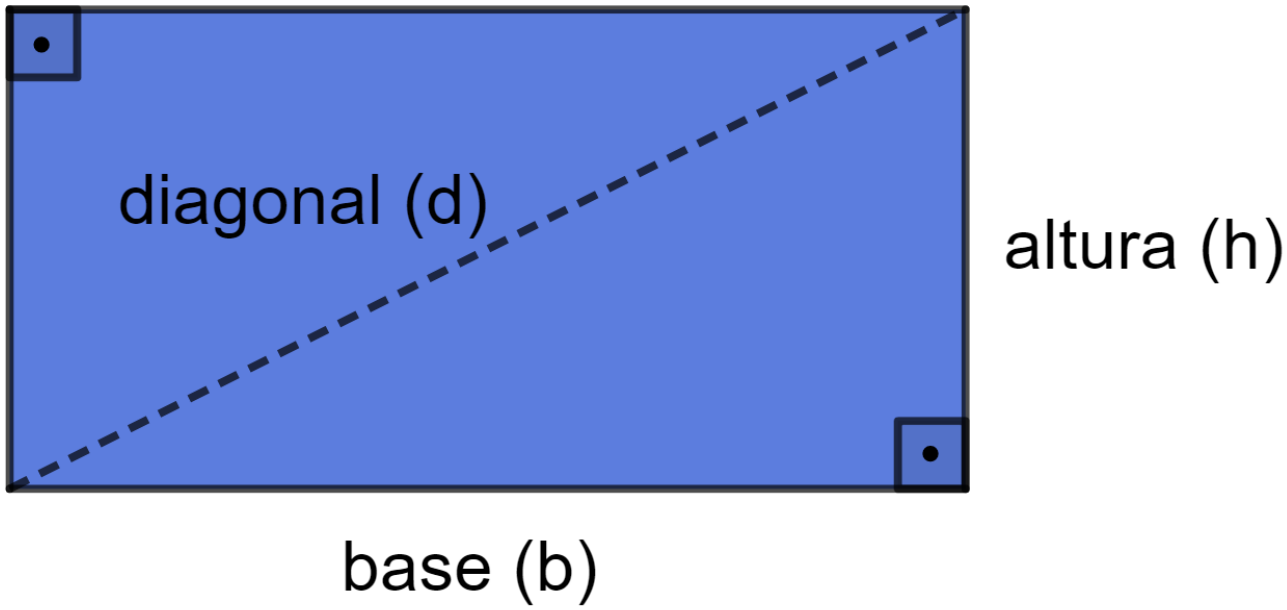
# POO e Javascript no backend

- 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

```
module.exports = class Retangulo {
    constructor() {
        //aqui são definidos todos os atributos da classe
        //observe que todos os atributos sempre começam com _
        //começa com _ e letra minuscula
        this._base = 0;
        this._altura = 0;
    }

    calcularArea() {
        return this._base * this._altura;
    }

    calcularPerimetro() {
        return 2 * (this._base*1 + this._altura*1);
    }

    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;
    }
}
```

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

```
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 {
    constructor() {
        //aqui são definidos todos os atributos da classe
        //observe que todos os atributos sempre começam com _
        //começa com _ e letra minúscula
        this._base = 0;
        this._altura = 0;
    }

    calcularArea() {
        return this._base * this._altura;
    }

    calcularPerimetro() {
        return 2 * (this._base*1 + this._altura*1);
    }

    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

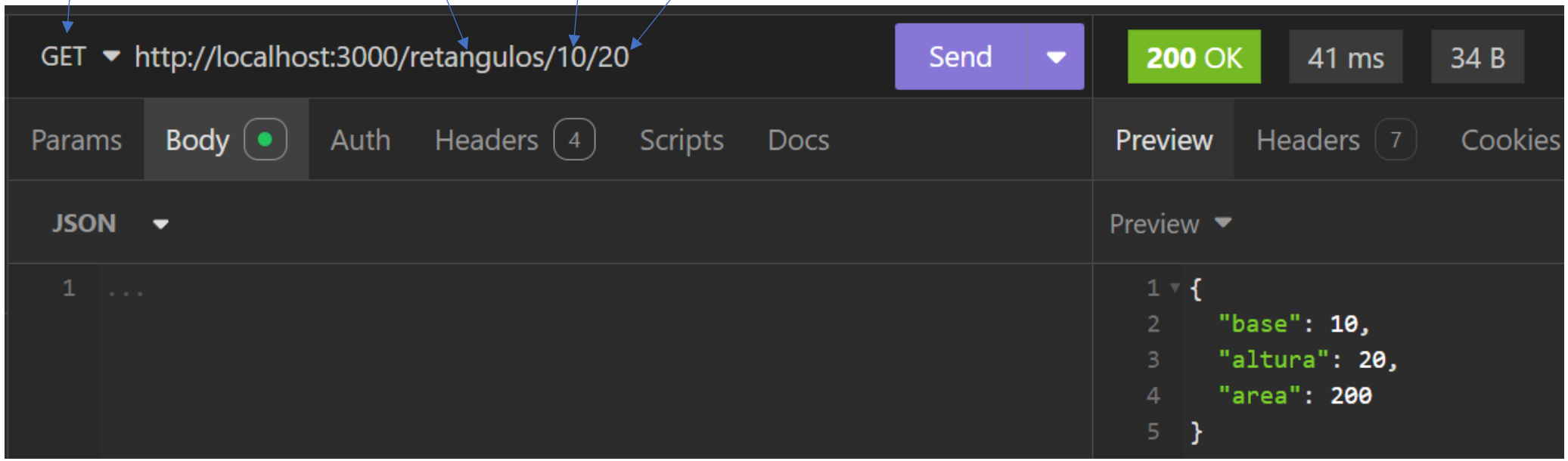
Get/set

```
?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) => {

});
```



# MVC

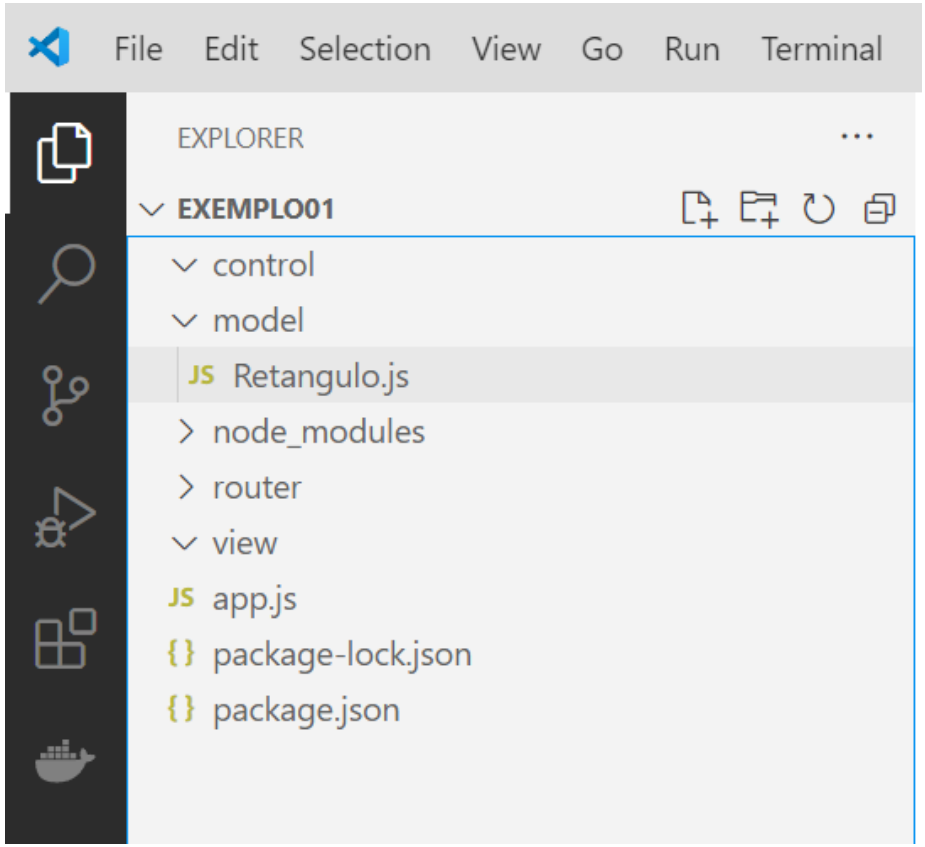
- O padrão MVC (Model-View-Controller) é uma arquitetura de software que separa a aplicação em três componentes principais:
  - Model (Modelo):Classes
    - Responsável pela lógica de negócios e pela interação com o banco de dados.
    - Contém os dados da aplicação e as regras de negócio.
    - Notifica 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);

});
```

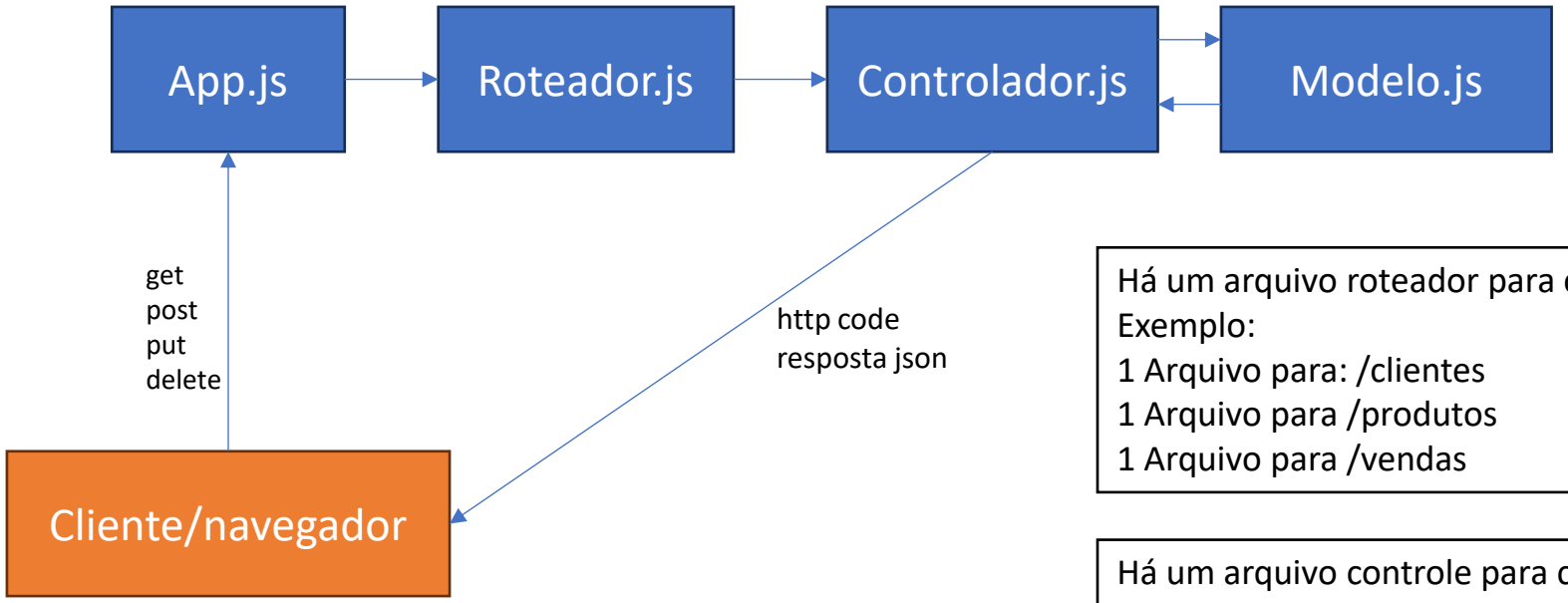
- O roteamento e controle estão no mesmo arquivo, é comum em uma aplicação Javascript separar o roteamento do controle.

# Arquitetura de pastas




- Construa uma pasta para cada funcionalidade do sistema
  - Model
  - View
  - Control
  - Router //para o roteamento

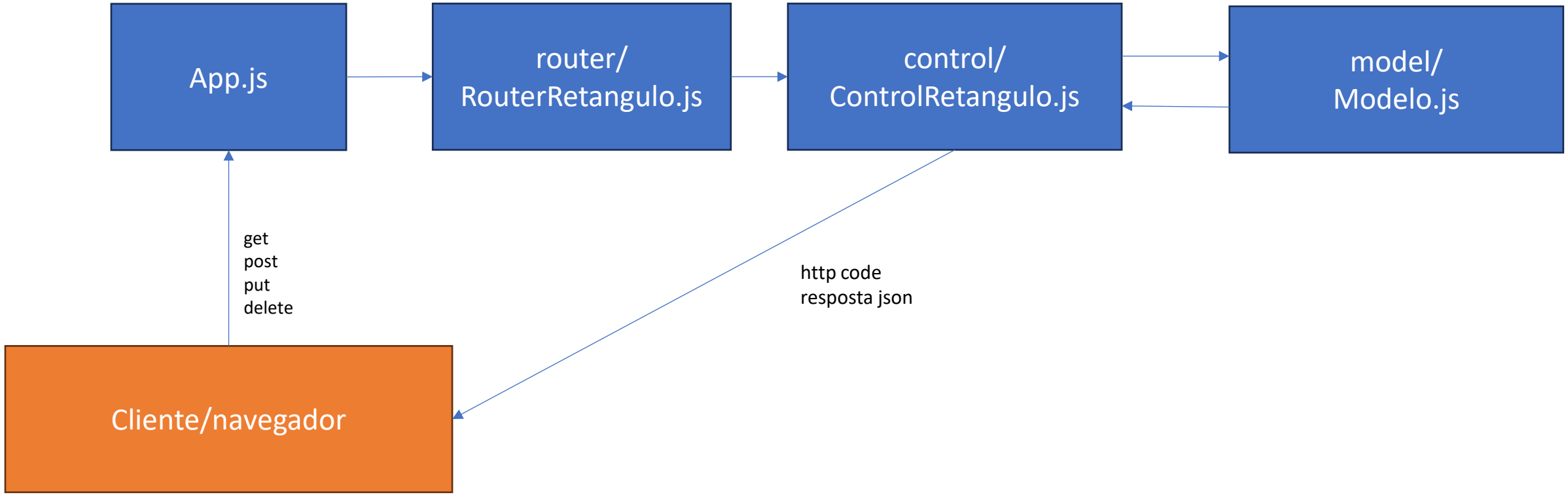
# Arquitetura básica:




Há um arquivo roteador para cada rota base
Exemplo:
1 Arquivo para: /clientes
1 Arquivo para /produtos
1 Arquivo para /vendas

Há um arquivo controle para cada modelo.
Exemplo:
O modelo Cliente tem um único Arquivo controle que trata o create, read, update, delete.

# Arquitetura básica na prática:



# App.js

```
const express = require('express'); // Importa o Express
// importa a classe de roteamento do retangulo
const RouterRetangulo = require('./router/RouterRetangulo');
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
const roteadorRetangulo = new RouterRetangulo();
//caso chegue uma requisição para /retangulos
//a classe  RouterRetangulo.js irá fazer o tratamento.
app.use('/retangulos',
    //método que trata as requisições http para /retangulos
    roteadorRetangulo.criarRotasRetangulo()
)
// Inicia a espera por requisições HTTP
app.listen(portaServico, () => {
    console.log(`API rodando no endereço: http://localhost:${portaServico}/`);
});
```

# RouterRetangulo.js

```
//importa o do express
const express = require('express');
//importa a classe de controle de retangulo
const ControlRetangulo = require('../control/ControlRetangulo');

module.exports = class RouterRetangulo {
    //cria o método responsavel por tratar a rotas
    //de /retangulos
     criarRotasRetangulo() {
        this._router = express.Router();
        this._controleRetangulo = new ControlRetangulo();
        //trata a rota GET: /retangulos/:base/:altura
        this._router.get('/:base/:altura',
            this._controleRetangulo.controle_get_calcular_tudo
        );
        //crie um método para cada roda...
        this._router.post('/:base/:altura',
            //chame o controle correspondente
        );
        return this._router;
    }
}
```

# ControlRetangulo.js

```
const express = require('express');
const Retangulo = require('./../model/Retangulo');
module.exports = class ControlRetangulo {
    //crie um controle para cada rota
    //para cada rota haverá um controle
    //método controle da rota /retangulos/:base/:altura
    controle_get_calcular_tudo(request, response) {
        //recupera a base passada pela uri
        const base = request.params.base;
        //recupera a altura passada pela uri
        const altura = request.params.altura;
        //faz uma instancia da classe retangulo
        const retangulo1 = new Retangulo();
        retangulo1.base = base;
        retangulo1.altura = altura;
        const objResposta = {
            perimetro: retangulo1.calcularPerimetro(),
            diagonal: retangulo1.calcularDiagonal(),
            area: retangulo1.calcularArea()
        }
        response.status(200).send(objResposta);
    }
}
```

# Retangulo.js

```
module.exports = class Retangulo {
    constructor() {
        //aqui são definidos todos os atributos da classe
        //observe que todos os atributos sempre começam com _
        //começa com _ e letra minúscula
        this._base = 0;
        this._altura = 0;
    }
    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;
    }
}
```

# 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.

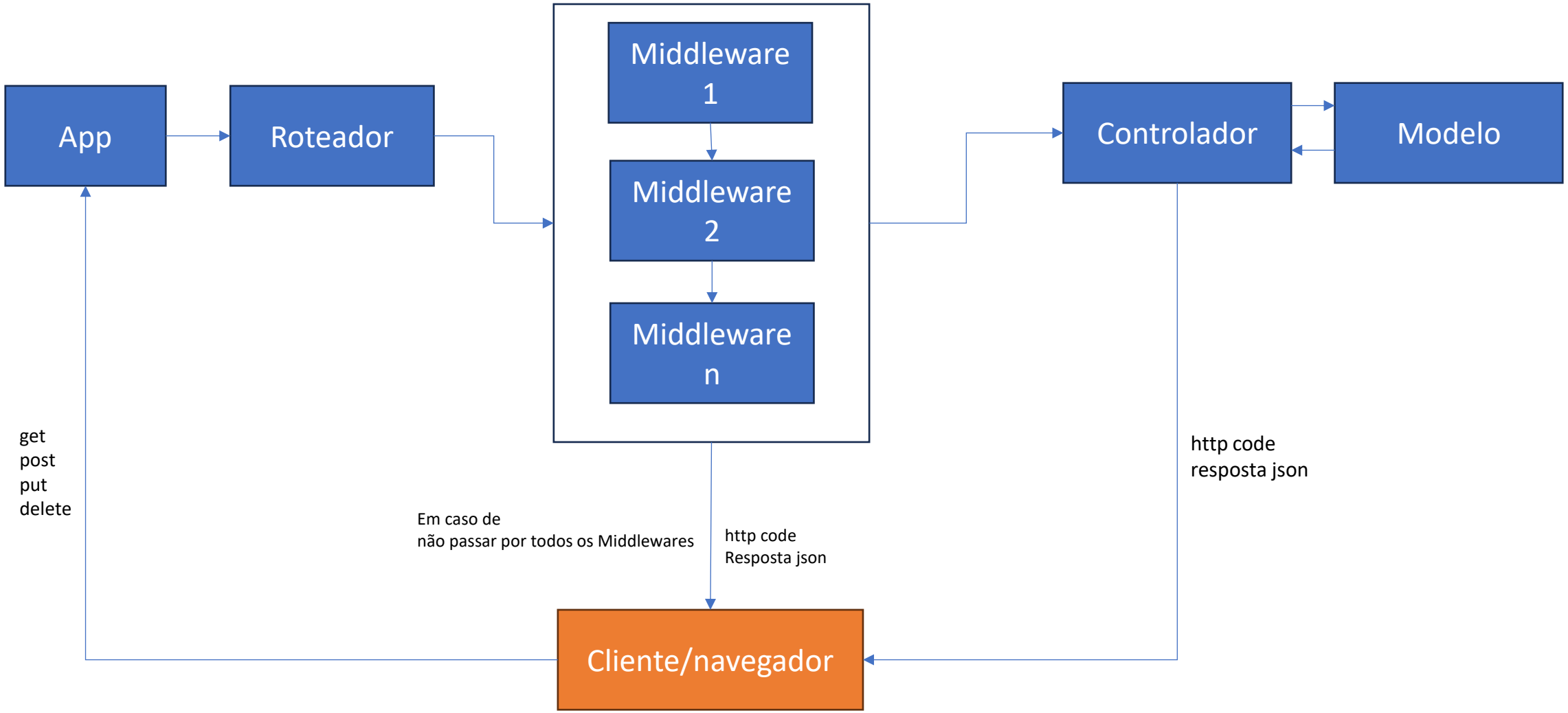
# Para Que Serve o Middleware?

- Midleware 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**.

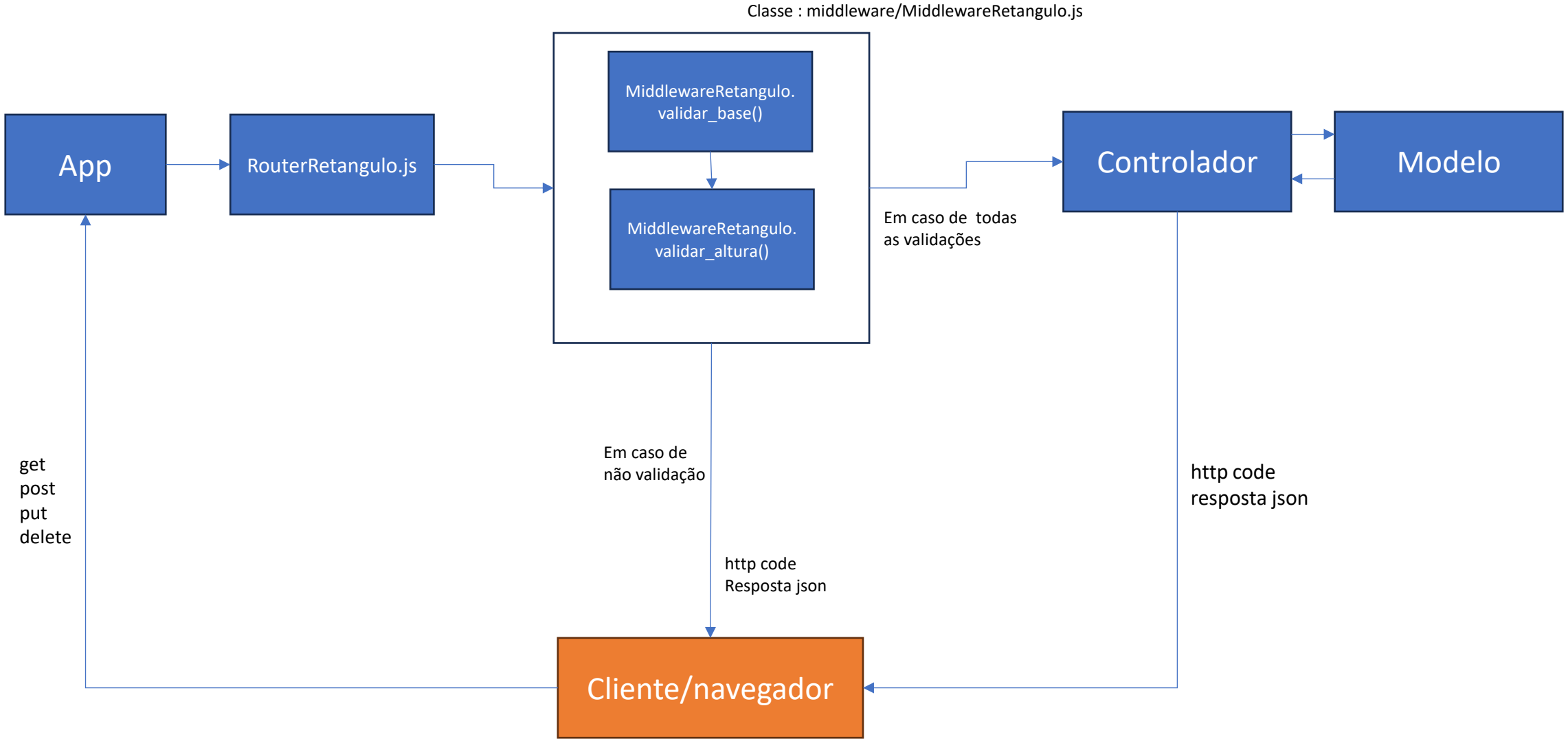
# Qual seria o Middleware para o exemplo do Retângulo?

- 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 especifico.

# Arquitetura básica:



# Arquitetura Retângulo:



RouterRetangulo.js

```
//importa o do express
const express = require('express');
//importa a classe de controle de retangulo
const ControlRetangulo = require('../control/ControlRetangulo');
const MiddlewareRetangulo = require('../middleware/middlewareRetangulo');

module.exports = class RouterRetangulo {
    //cria o método responsavel por tratar todas as rotas
    //de /retangulos
     criarRotasRetangulo() {
        this._router = express.Router();
        this._controleRetangulo = new ControlRetangulo();
        this._middlewareRetangulo = new MiddlewareRetangulo();
        //trata a rota GET: /retangulos/:base/:altura
        this._router.get('/:base/:altura',
            this._middlewareRetangulo.validar_base, // se valido: next() vai pro proximo
            this._middlewareRetangulo.validar_altura, // se valido: next() vai pro proximo
            this._controleRetangulo.controle_get_calcular_tudo
        );
        //crie um método para cada roda...
        this._router.post('/:base/:altura',
            //chame o controle correspondente
        );
        return this._router;
    }
}
```

# MiddlewareRetangulo.js

```
const express = require('express');
const Retangulo = require('../model/Retangulo');
module.exports = class MiddlewareRetangulo {

    validar_base(request, response, next) {
        //recupera a base passada pela uri
        const base = request.params.base;
        if (isNaN(base)) {
            const objResposta = {
                status: false,
                msg: "A base deve ser um número!"
            }
            response.status(200).send(objResposta);
        } else {
            next(); // Chama o próximo middleware ou rota
        }
    }
    validar_altura(request, response, next) {
        //recupera a base passada pela uri
        const altura = request.params.altura;
        if (isNaN(altura)) {
            const objResposta = {
                status: false,
                msg: "A altura deve ser um número!"
            }
        } else {
            next(); // Chama o próximo middleware ou rota
        }
    }
}
```