API REST DE ACESSO A BANCO DE DADOS

Prof. Me. Hélio Esperidião

A origem do termo REST (Rest API)

- Os conceitos do REST foram estabelecidos na tese de doutorado de *Roy Fielding* nos anos 2000, onde o princípio fundamental é utilizar o protocolo HTTP para a comunicação de dados.

API do Tipo REST

- A sigla REST refere-se a Representational State Transfer (Transferência de Estado Representacional) e é um estilo de arquitetura de software.
- Uma API REST, portanto, implica em um **conjunto de restrições que devem ser seguidas durante o desenvolvimento de uma aplicação na internet**.
- Essas restrições permitem a criação de uma aplicação com uma interface bem definida, com rotinas padronizadas e facilmente representáveis, o que facilita a comunicação entre cliente e servidor.

arquitetura REST

- É simples e oferece acesso aos recursos para que o cliente possa acessar e renderizar esses recursos.
- No estilo REST, as URI (Uniform Resource Identifier) ou são usadas para identificar cada recurso. (**ROTA DOS RECURSOS**)

- Essa arquitetura utiliza várias representações de recursos para expressar o seu tipo, como XML, **JSON**, texto, imagens, entre outros. (**JSON muito utilizado atualmente**)

Responsabilidades no REST

- No contexto do REST, existe um princípio chamado "**statelessness**" (sem estado), onde o servidor não precisa estar ciente do estado do cliente e vice-versa. Mas o que exatamente são o servidor e o cliente?
- **Cliente**: É o componente que solicita um serviço e envia requisições para diferentes tipos de serviços ao servidor.
- **Servidor**: É o componente que fornece serviços e continua a atender as solicitações do cliente.

REST

- A separação de responsabilidades entre a interface do usuário e o armazenamento de dados é facilitada pela relação cliente-servidor.
- Quando uma solicitação REST é feita, o servidor envia uma representação do estado solicitado.
- Não há limite máximo para o número de clientes que podem ser atendidos por um único servidor.
- Não é necessário que o cliente e o servidor residam em sistemas juntos.
- A comunicação entre o cliente e o servidor ocorre por meio da troca de mensagens usando um padrão de **solicitação-resposta**.
- O cliente envia uma solicitação de serviço e o servidor retorna uma resposta correspondente.

Requisições e comunicações

- O REST precisa que um cliente faça uma requisição para o servidor para enviar ou modificar dados. Um requisição consiste em:

- **Método HTTP**: Define que tipo de operação o servidor vai realizar;
- **Header:** deve conter o cabeçalho da requisição que passa informações sobre a requisição;
- **Rota**: para o servidor, como por exemplo https://helioesperidiao/alunos/;
- Informação no corpo da requisição, sendo esta informação opcional.

Método HTTP

- Em aplicação REST, os métodos mais utilizados são:
  - **GET**: é o método mais comum, geralmente é usado para solicitar que um servidor envie um recurso;
  - **POST:** foi projetado para enviar dados de **entrada** para o servidor. Na prática, é frequentemente usado para suportar formulários HTML;
  - **PUT:** Edita ou atualiza documentos em um servidor;
  - **DELETE** : Apaga dado ou coleção de dados do servidor
  - **PATCH**: Atualizar parcialmente um determinado recurso.
  - **HEAD**: Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si.
  - **OPTIONS**: Obter quais manipulações podem ser realizadas em um determinado recurso.
  - TRACE :Devolve a mesma requisição que for enviada veja se houve mudança e/ou adições feitas por servidores intermediários.
  - CONNECT: Converte a requisição de conexão para um túnel TCP/IP transparente, geralmente para facilitar a comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado.

URL – Uniform Resource Locator

- Localizador de Recursos Universal se refere ao local, o Host que você quer acessar determinado recurso.
- O objetivo da URL é associar um endereço remoto com um nome de recurso na Internet.
- Exemplo de URL
  - http://helioesperidiao.com
  - http://univap.br
  - http://google.com.br
- Acessando esses endereços você cai no servidor onde está minha página

URN – Uniform Resource Name

- **Nome de Recursos Universal** é o nome do recurso que será acessado e também fará parte da URI.

- login.html
- inicio.php
- contato.html
- É comum associarmos URN a página(Arquivo) que estamos acessando.

- Outro ex.: /api/v01/usuarios/

URI = URL+ URN

- A URI une o Protocolo (http://) a localização do recurso (URL - app.com.br) e o nome do recurso (URN - /alunos/1b/) para que você acesse as coisas na Web.

- URI = http://app.com.br/alunos/1b

URI - Uniform Resource Identifier

- Identificador de Recursos Universal é o identificador do recurso.
- Pode ser uma imagem, uma página, etc, pois tudo o que está disponível na internet precisa de um identificador único para que não seja confundido.
- Exemplos de URI
  - http://www.app.com/alunos

Exemplo de uso

- GET http://www.app.com/alunos
- DELETE http://www. app.com/alunos/27
- POST http://www. app.com/alunos –data {nome: Maria Joana}

Exemplos de URI

- GET
  - http://api.com.br/clientes
  - Recuperar os dados de todos os clientes.
- GET
  - http://api.com.br/clientes/id
  - Recuperar os dados de um determinado cliente.
- POST
  - http://api.com.br/clientes
  - Criar um novo cliente.
- PUT
  - http://api.com.br/clientes/id
  - Atualizar os dados de um determinado cliente.
- DELETE
  - http://api.com.br/clientes/id

Códigos de Respostas

- **200 (OK):** requisição atendida com sucesso;
- **201 (CREATED)**: objeto ou recurso criado com sucesso;
- **204 (NO CONTENT):** objeto ou recurso deletado com sucesso;
- **400 (BAD REQUEST):** ocorreu algum erro na requisição (podem existir inúmeras causas como erro ou falta de parâmetros no corpo da requisição);
- **404 (NOT FOUND):** rota ou coleção não encontrada;
- **500 (INTERNAL SERVER ERROR):** ocorreu algum erro no servidor.

Códigos de Respostas

- **401 Unauthorized**: Em requisições que exigem autenticação, mas seus dados não foram fornecidos.
- **403 Forbidden**: Em requisições que o cliente não tem permissão de acesso ao recurso solicitado.
- **429 Too Many Requests**: No caso do serviço ter um limite de requisições que pode ser feita por um cliente, e ele já tiver sido atingido.
- **503 Service Unavailable**: Em requisições feitas a um serviço que esta fora do ar, para manutenção ou sobrecarga.

E qual a diferença entre **REST** e **RESTful**?

- A API REST é um conjunto de boas práticas e um modelo de **arquitetura de software** que estabelece uma série de requisitos para o desenvolvimento de APIs
- O protocolo HTTP é o principal pilar sobre o qual toda a arquitetura se baseia.

REST vs RESTful

- REST: conjunto de princípios de arquitetura
- RESTful: capacidade de determinado sistema aplicar os princípios de REST.

RESTful: Requisitos

- Para que a arquitetura de um sistema seja considerada RESTful, é necessário possuir certos padrões:
- **cliente-servidor:**
  - Arquitetura que deve utilizar como padrão de comunicação o HTTP;
- **interface uniforme:**
  - descreve a integração uniforme entre cliente e servidor, construindo uma interface que identifique e represente recursos bem definidos, mensagens autodescritivas;
- **stateless:**
  - aponta que cada interação via API tem acesso a dados completos;
- **cache**:
  - indispensável para reduzir o tempo médio de resposta, melhorando a eficiência, desempenho e escalabilidade da comunicação;
- **camadas**:
  - concede que a solução seja menos complexa, altamente flexível e desacoplada. Exemplo: MVC

A maturidade de Richardson

- Leonard Richardson fez uma análise em várias APIs diferentes e criou Modelo de Maturidade.
- Para que uma API seja considerada RESTful ela deve satisfazer os 4 pilares (Nível 0, 1, 2 e 3).

Nível 0 ou POX

- A API deve utilizar o protocolo **HTTP** para a comunicação.
- Esse é considerado o nível mais básico e uma API que implementa apenas esse nível **não pode ser considerada REST**.
- Nesse nível os nomes dos recursos não seguem qualquer padrão e estão sendo usados apenas para fazer invocação de métodos remotos.
- Nesse nível usamos o protocolo HTTP para comunicação, mas sem seguir qualquer tipo de regras para implementar os métodos.

Nível 0 — POX

Verbo HTTP	URI	Operação
GET	/getCliente/1	Pesquisar
POST	/salvarCliente	Criar
POST	/alterarCliente/1	Alterar
GET/POST	/excluirCliente/1	Excluir

NÍVEL 1 – RECURSOS (RESOURCES)

- A API deve possuir mapeamento de recursos bem definidos.
- Os recursos devem ser Representados por substantivos e também faz a correta utilização de URIs para cada recurso respectivamente.
- Nesse nível representamos cada recurso fazemos por substantivos no plural.
  - No nível 1 já usamos os verbos HTTP de forma correta, já que se os verbos não fossem usados, as rotas de pesquisar, alterar e incluir **ficariam ambíguas**.

Nível 1

- No nível 1 já usamos os verbos HTTP de forma parcialmente correta, já que se os verbos não fossem usados, as rotas de pesquisar, alterar e incluir **ficariam ambíguas**.
- No nível 1 poderíamos utilizar POST no lugar do GET. Observe que são URIs Diferentes então poderíamos utilizar.

Verbo HTTP	URI	Operação
GET	/clientes/1	Pesquisar
POST	/clientes	Criar
PUT	/clientes/1	Alterar
DELETE	/clientes/1	Excluir

semântica

- Em programação, a semântica se refere ao significado e interpretação do código.
- É a maneira como as instruções e expressões em um programa são entendidas e executadas pelo computador.
- A semântica está relacionada ao comportamento e à lógica do programa, envolvendo as regras e as operações definidas pela linguagem de programação.

semântica

- A semântica determina como as instruções e expressões são executadas, como os valores são atribuídos, como as estruturas de controle são avaliadas e como as funções são chamadas.
- Ela define o propósito e a intenção de um trecho de código e como esse código interage com outros componentes do programa.

Nível 2

- A API deve utilizar o protocolo HTTP de forma **semântica** com seus verbos, fazendo o uso do GET, POST, PUT, DELETE, etc de acordo com o tipo de requisição e também deve possuir os tipos corretos de retornos para cada resposta possível.
- **É importante o retorno correto dos status codes de cada endpoint após cada operação.(200 –OK, etc)**

Nível 2

- Semanticamente seria incorreto trocar GET por POST, apesar de poder substituir e não causar ambiguidade de recursos semanticamente o get é para “recuperar” e o post para “Inserir”.
- No nível 2 a semântica é importante.

Verbo HTTP	URI	Operação
GET	/clientes/1	Pesquisar
POST	/clientes	Criar
PUT	/clientes/1	Alterar
DELETE	/clientes/1	Excluir

Nível 3

- A API deve fazer o uso de HATEOS (Hipermídeas), mostrando seu estado atual e também o relacionamento com os demais recursos presentes na API.

hypermedia

- Repare que customer possui hypermedia (links) que apontam para ele mesmo (estado atual) e delete (estado futuro).
- O ponto mais importante do hypermedia controls é a maneira que um resource deve ser manipulado é descrito, não tendo necessidade de adivinhações de quais operações estão de fato implementadas.

```
GET /clientes/1
{
   “nome": “Helio",
    “nascimento”: “1985/10/03”,
   "links": [ {
      "rel": “GET",
      "href": "http://localhost:8080/clientes/1"
   }, {
      "rel" : “DELETE",
      "href": "http://localhost:8080/clientes/1"
   ]
}
```

GLÓRIA DO RESTFUL

- Se a API de sua aplicação ou micro serviço satisfaz todos esses níveis, ela sim pode ser considerada uma **API RESTful**!

- Esse modelo pode ser utilizado como guia para squads que buscam melhorar o ecossistema interno da companhia. Essa classificação colabora com o processo de melhoria contínua dos times e do profissional.

Exemplo Completo CRUD Clientes

Sql injection

- é uma técnica de ataque que envolve a manipulação do código SQL.
- SQL Injection é uma classe de ataque onde o invasor pode inserir ou manipular consultas criadas pela aplicação, que são enviadas diretamente para o banco de dados relacional.
- Por que o SQL Injection funciona?
  - A aplicação aceita dados fornecidos pelo usuário;
    - Você pede para digitar o nome, mas quem garante que ele não digite código sql de forma maliciosa?

Sql injection na prática.

- Imagine o seguinte algoritmo:

$nome = $_GET[‘nome’];
$sql = “select * from Cliente Where nome = ‘$nome’”;

Se o usuário usar digitar o valor da variável $nome for igual a:Helio temos:
$sql = select * from Cliente Where nome = ‘helio’

Sql injection na prática.

- O problema é quando o usuário não digita o nome;
- Imagine que no lugar do nome o usuário digite: `' OR 1=1; #'`

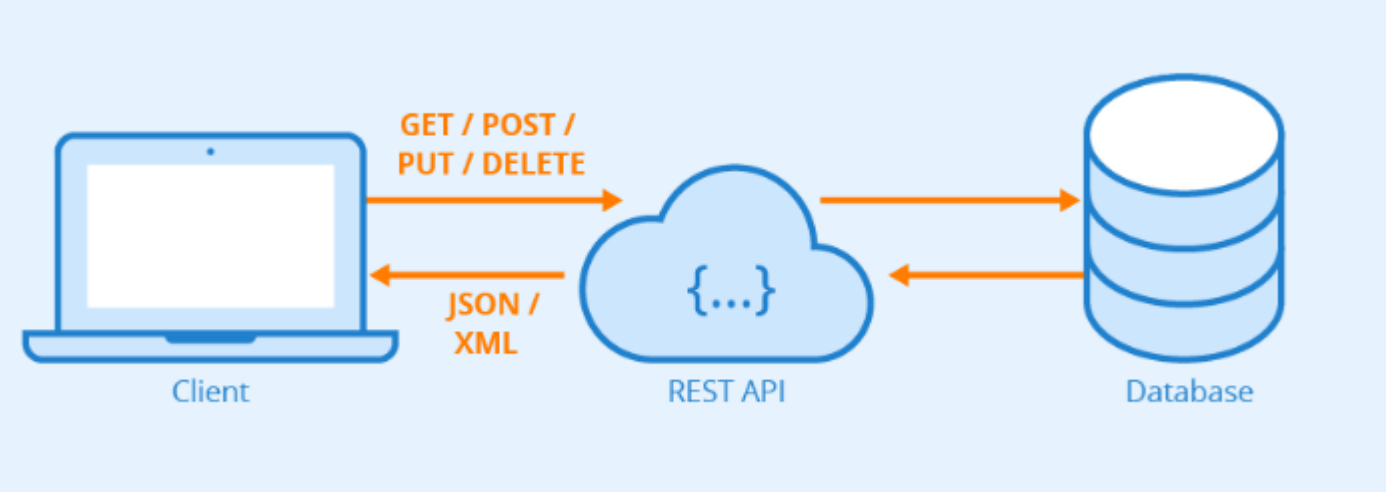
$nome = $_GET[‘nome’];
$sql = “select * from Cliente Where nome = ‘$nome’”;

- Teríamos algo assim:

```
SELECT * FROM clientes WHERE nome ='' OR 1=1; #’
```

- *Depois do # é comentário, e serão apresentados todos os clientes.*
- *Esse é um exemplo simples, mas o usuário pode criar instruções que podem potencialmente excluir o banco de dados.*

Arquitetura



Arquitetura de pastas e arquivos.

- .htaccess
- Index.php
- Controle
  - Cliente_cadastrar.php
  - Cliente_excluir.php
  - Cliente_buscar.php
  - Cliente_listar.php
  - Cliente_atualizar.php
- Modelo
  - Banco.php
  - Cliente.php
- Visualizacao
  - Cadastrar.html

Rotas e recursos

VERBOS	URI(ROTA)	OPERAÇÃO	ARQUIVO RESPONSÁVEL
GET	http://localhost/clientes/	Retorna todos os clientes.	controle/Cliente_listar.php
GET	http://localhost/clientes/id	Retorna um cliente pelo id	controle/Cliente_buscar.php
POST	http://localhost/clientes	Cadastra um novo cliente	controle/Cliente_cadastrar.php
PUT	http://localhost/clientes/id	Atualiza um cliente pelo seu id.	Controle/Cliente_atualizar.php
DELETE	http://localhost/clientes/id	Exclui um cliente.	controle/Cliente_excluir.php

.htaccess

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ /index.php?path=$1 [NC,L,QSA]
```

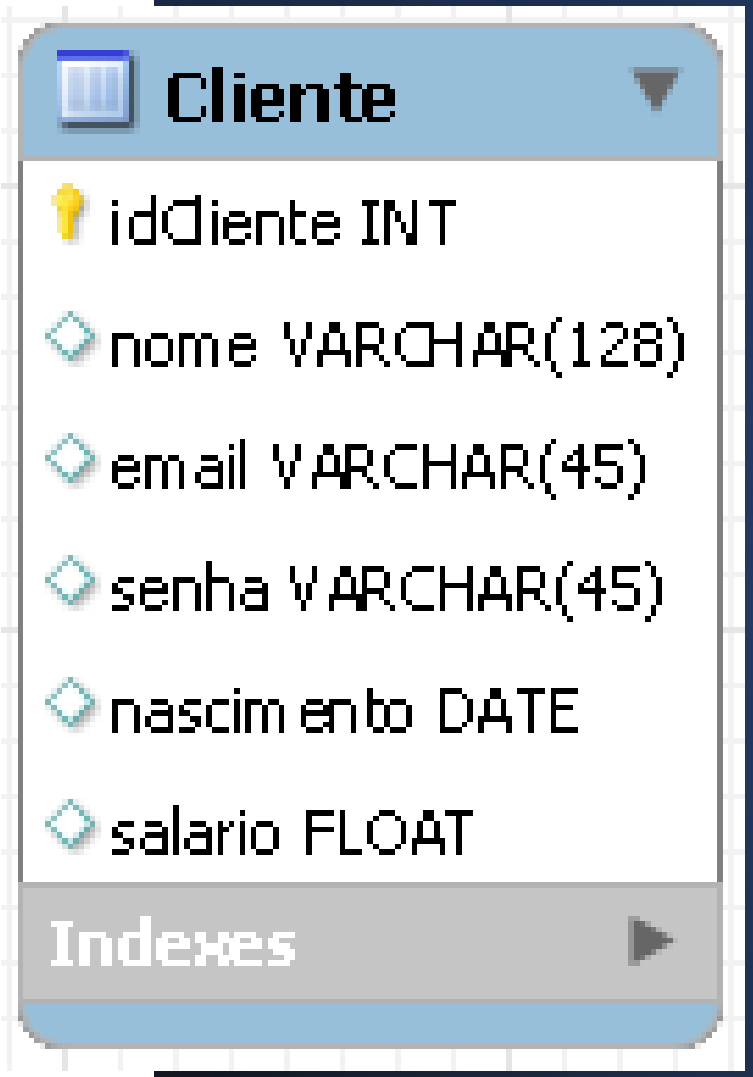
Index.php (roteador)

```
$metodo = $_SERVER['REQUEST_METHOD'];
  $partesRota = explode("/", $_SERVER['REQUEST_URI']);
  if($partesRota[1]=="clientes"){
     if( $metodo=="POST"){
        require_once "controle/Cliente_cadastrar.php";
     }elseif($metodo=="PUT"){
        require_once "controle/Cliente_atualizar.php";
     }elseif($metodo=="DELETE"){
        require_once "controle/Cliente_excluir.php";
     }elseif($metodo=="GET"){
        if(isset($partesRota[2])){
           if(is_numeric($partesRota[2])){
              require_once "controle/Cliente_buscar.php";
           }else if ($partesRota[2]==""){
              require_once "controle/Cliente_listar.php";
           }
        }else{
           require_once "controle/Cliente_listar.php";
        }
     }else{
        header("HTTP/1.1 404 Not Found");
     }
  }else{
     header("HTTP/1.1 404 Not Found");
  }
```

VERBOS	URI(ROTA)	ARQUIVO RESPONSÁVEL
GET	/clientes/	controle/Cliente_listar.php
GET	/clientes/id	controle/Cliente_buscar.php
POST	/clientes	controle/Cliente_cadastrar.php
PUT	/clientes/id	Controle/Cliente_atualizar.php
DELETE	/clientes/id	controle/Cliente_excluir.php

Tabela Cliente

```
CREATE SCHEMA IF NOT EXISTS aulaPHPmysql;
USE aulaPHPmysql;

CREATE TABLE IF NOT EXISTS aulaPHPmysql.Cliente (
  idCliente INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(128),
  email VARCHAR(45),
  senha VARCHAR(45),
  nascimento DATE,
  salario FLOAT
);
```



Banco.php

```
class Banco{
    private $host     = "127.0.0.1";
    private $usuario  = "root";
    private $senha    = "";
    private $banco    = "aulaphpmysql";
    private $porta     = "3306";
    private $con=null;
    private function conectar(){
        $this->con = new mysqli( $this->host, $this->usuario,$this->senha,$this->banco, $this->porta);
        if ( $this->con->connect_error) {
            die("Falha ao conectar: " .  $this->con->connect_error);
        }
    }
    public function getConexao(){
        if( $this->con==null){
            $this->conectar();
        }
        return $this->con;
    }
}
```

Atributos da classe de conexão com o banco

Executa quando o método getConexao() é chamado
e o atributo $con é igual a null
Método privado é executado somente dentro da instância.

Cria uma conexão com o banco.
A conexão é armazenada no atributo privado $con

Método que retorna a conexão uma conexão com o banco.
Sempre é chamado quando se deseja executar código sql

Cliente.php (1/8) - Atributos

```
<?php
include "Banco.php";
class Cliente implements JsonSerializable
{
    private $idCliente;
    private $nome;
    private $email;
    private $senha;
    private $nascimento;
    private $salario;
    private $banco;
```

Inclui o arquivo contendo a classe Banco

Atributos da classe Cliente, observe que os atributos correspondem aos campos da tabela cliente

Cliente.php (2/8) – jsonSerialize

É chamado quando a instancia do objeto é convertida em json.
Observe que não é interessante enviar para o cliente a senha do usuário

```
public function jsonSerialize() {
    $json= array();
    $json['idCliente']=$this->getIdCliente();
    $json['nome']=$this->getNome();
    $json['email']=$this->getEmail();
    $json['nascimento']=$this->getNascimento();
    $json['salario']=$this->getSalario();
    return $json;
}
```

Cliente.php (3/8) - cadastrar()

Por meio da instancia da classe banco recupera a conexão. Tento a conexão é possível executar comando sql no sgbd

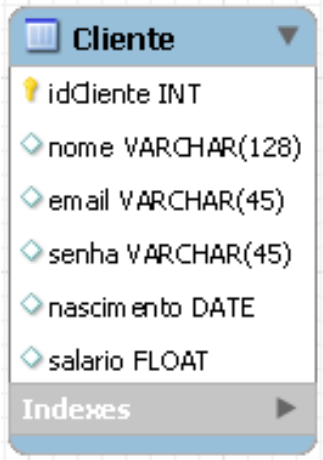
Prepara a instrução sql, posteriormente os "**?**" serão substituídos por valores. Método impede um ataque comum na web chamado de ***sql injection***

```
public function cadastrar()
  {
     $this->banco = new Banco();
     $stmt = $this->banco->getConexao()->prepare("insert into Cliente (nome, email,senha,nascimento,salario )values(?,?,?,?,?)");
     $stmt->bind_param("sssss", $this->nome, $this->email, $this->senha, $this->nascimento, $this->salario);
     $resposta = $stmt->execute();
     $idCadastrado = $this->banco->getConexao()->insert_id;
     $this->setIdCliente($idCadastrado);
     return $resposta;
  }
```

Substitui os ? Pelos valores das variáveis

- i - integer
- d - double
- s - string

s – String – nome
s – String – email
s – String - nascimento
d – Double - salario



Cliente.php (4/8) - excluir()

```
public function excluir(){
    $stmt = $this->banco->getConexao()->prepare("delete from Cliente where idCliente = ?");
    $stmt->bind_param("i", $this->idCliente);
    return $stmt->execute();
}
```

Executa o código sql

- i - integer
- d - double
- s - string

Substitui os ? Pelos valores das variáveis

Cliente.php (5/8) - atualizar()

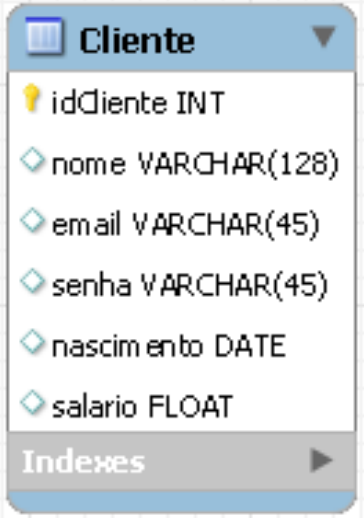

```
public function atualizar(){
    $stmt =$this->banco->getConexao()->prepare("update cliente
                    set nome=?,
                    email=?,
                    senha=?,
                    salario=?,
                    nascimento=?
                    where idCliente = ?");
    $stmt->bind_param("sssdsi", $this->nome,$this->email,$this->senha,$this->salario,$this->nascimento,$this->idCliente);
    $stmt->execute();
}
```

- i - integer
- d - double
- s - string

Substitui os ? Pelos valores das variáveis

Cliente.php (6/8) - buscar ($idCliente)

```
public function buscar($idCliente)    {
    $this->banco = new Banco();
    $stmt = $this->banco->getConexao()->prepare("select * from cliente where idCliente = ?");
    $stmt->bind_param("i", $idCliente);
    $stmt->execute();
    $resultado = $stmt->get_result();
    while ($linha = $resultado->fetch_object()) {
        $this->setIdCliente($linha->idCliente);
        $this->setNome($linha->nome);
        $this->setEmail($linha->email);
        $this->setSenha($linha->senha);
        $this->setNascimento($linha->nascimento);
        $this->setSalario($linha->salario);
    }
    return $this;
}
```



Passar por todas as linhas da matriz Resultado como essa matriz possui apenas uma linha executa uma única vez.

Cliente.php (7/8) - listar()

```
public function listar()    {
        $this->banco = new Banco();
        $stmt = $this->banco->getConexao()->prepare("Select * from Cliente");
        $stmt->execute();
        $result = $stmt->get_result();
        $vetorClientes = array();
        $i = 0;
        while ($linha = mysqli_fetch_object($result)) {
            $vetorClientes[$i] = new Cliente();
            $vetorClientes[$i]->setIdCliente($linha->idCliente);
            $vetorClientes[$i]->setNome($linha->nome);
            $vetorClientes[$i]->setEmail($linha->email);
            $vetorClientes[$i]->setSenha($linha->senha);
            $vetorClientes[$i]->setNascimento($linha->nascimento);
            $vetorClientes[$i]->setSalario($linha->salario);
            $i++;
        }
        return $vetorClientes;
    }
```

Result é uma matriz com todos os dados da execução do sql

Cria um vetor para armazenar as instancias de Cliente

Passar por todas as linhas da matriz Result
Cria um vetor de objetos do tipo Cliente

Retorna um vetor com todos os clientes

Cliente.php (8/8) – getter & setter

```
 public function getIdCliente(){return  $this->idCliente;}
    public function setIdCliente($v){$this->idCliente=$v;}
    public function getNome(){return $this->nome;}
    public function setNome($nome){$this->nome = $nome;}
    public function getEmail(){return $this->email;}
    public function setEmail($v){$this->email=$v;}
    public function getSenha(){return $this->senha;}
    public function setSenha($v){$this->senha = $v;}
    public function getNascimento(){return $this->nascimento;}
    public function setNascimento($v){$this->nascimento=$v;}
    public function getSalario(){return $this->salario;}
    public function setSalario($v){$this->salario=$v;}

}
```

controle/Cliente_cadastrar.php - 1

```
include "modelo/Cliente.php";
// cria um array para utilizar na resposta.
$resposta = array();

//recupera os dados(texto json) que vieram no corpo da requisicao.
$jsonRecebido = file_get_contents('php://input');

//converte os dados em um objeto json
$obj = json_decode($jsonRecebido);

//strip_tags remove tags html
//dos dados  vindos dos clientes
//impede exemplo <b>helio</b>
$nome = strip_tags($obj->nome);
$email = strip_tags($obj->email);
$senha = strip_tags($obj->senha);
$salario = strip_tags($obj->salario);
$nascimento = strip_tags($obj->nascimento);
```

Como os dados são enviados dos clientes no corpo da requisição os vetores $_GET e $_POST não podem ser utilizados.
Para recuperar os dados no corpo da requisição utilize:
file_get_contents('php://input');

$obj é um objeto json criado a partir dos dados(texto json) que
vieram no corpo da requisição.

controle/Cliente_cadastrar.php - 2

```
//verifica se os dados foram preenchidos corretamente
//este exemplo é simples
//verifique conforme a necessidade da aplicação.
if($nome==""){
    $resposta['cod']="erro";
    $resposta['msg']="Nome não preenchido";

}else if($email==""){
    $resposta['cod']="erro";
    $resposta['msg']="E-mail não preenchido";
}else{
```

Verifique a integridade dos dados.
Faça todas as **condicionais** necessárias par impedir que sejam gravados dados inválidos ou inconsistentes.

controle/Cliente_cadastrar.php - 3

```
//após passar por todas as verificações
    $cliente = new Cliente();

    //passa os dados para o objeto
    $cliente->setNome($nome);
    $cliente->setEmail($email);
    $cliente->setSenha($senha);
    $cliente->setSalario($salario);
    $cliente->setNascimento($nascimento);

    //chama o método cadastrar
    $resultado = $cliente->cadastrar();
```

Crie uma instância da classe cliente preencha os atributos através dos métodos SET.
Ao final, após passar todos os dados para a instância da classe chame o método cadastrar();

controle/Cliente_cadastrar.php - 4

```
if($resultado==true){
        header('HTTP/1.1 201 Created');
        $resposta['cod']="ok";
        $resposta['msg']="cadastrado com sucesso";
        $resposta['POST']=$cliente;
    }else{
        header('HTTP/1.1 200 ok');
        $resposta['cod']="erro";
        $resposta['msg']="Não foi cadastrado";
    }
} if($resultado==true){
        header('HTTP/1.1 201 Created');
        $resposta['cod']="ok";
        $resposta['msg']="cadastrado com sucesso";
        $resposta['POST']=$cliente;
    }else{
        header('HTTP/1.1 200 ok');
        $resposta['cod']="erro";
        $resposta['msg']="Não foi cadastrado";
    }

}
```

Quando o método cadastrar() é executado e o cliente é inserido no banco de dados, o id do cliente inserido é recuperado e inserido nos atributos do objeto.
Verifique nos slides anteriores o método cadastrar() da classe Cliente.

Monte o vetor que será convertido em um JSON.
O Json será enviado como resposta.
Lembre-se de enviar os códigos HTTP Correspondentes

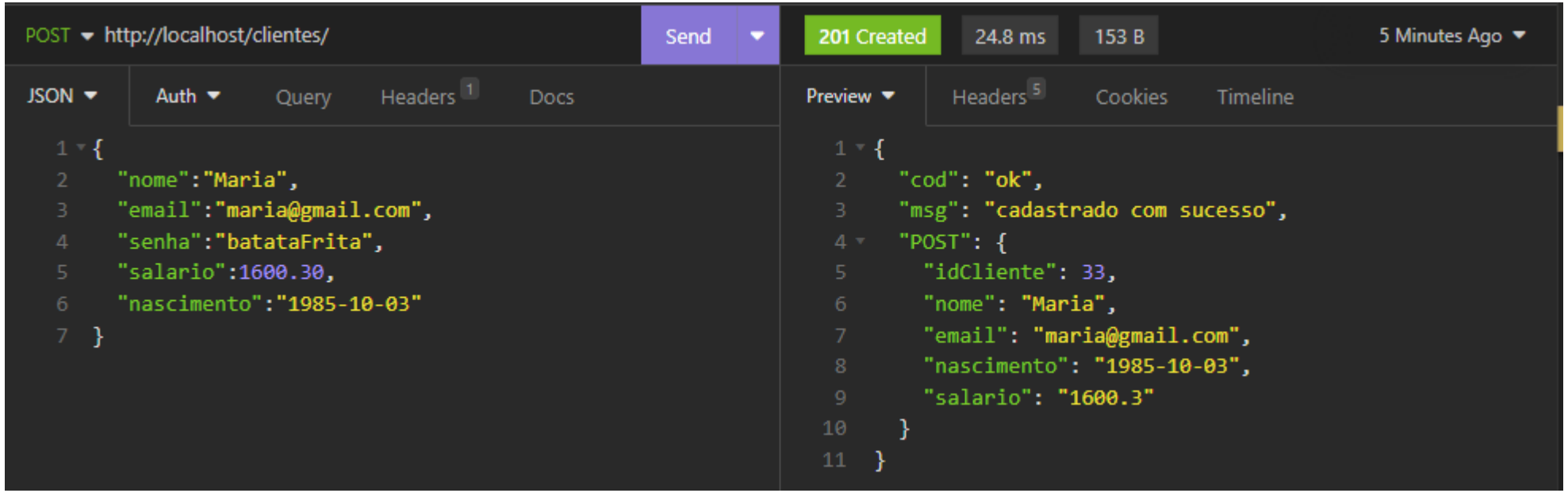
controle/Cliente_cadastrar.php - 5

```
//converte o array resposta em json para
//enviar de resposta ao cliente
echo json_encode($resposta);
?>
```

Converte o vetor $resposta em um json e escreve como resposta para o cliente.

Testar o controle
POST: http://localhost/clientes/

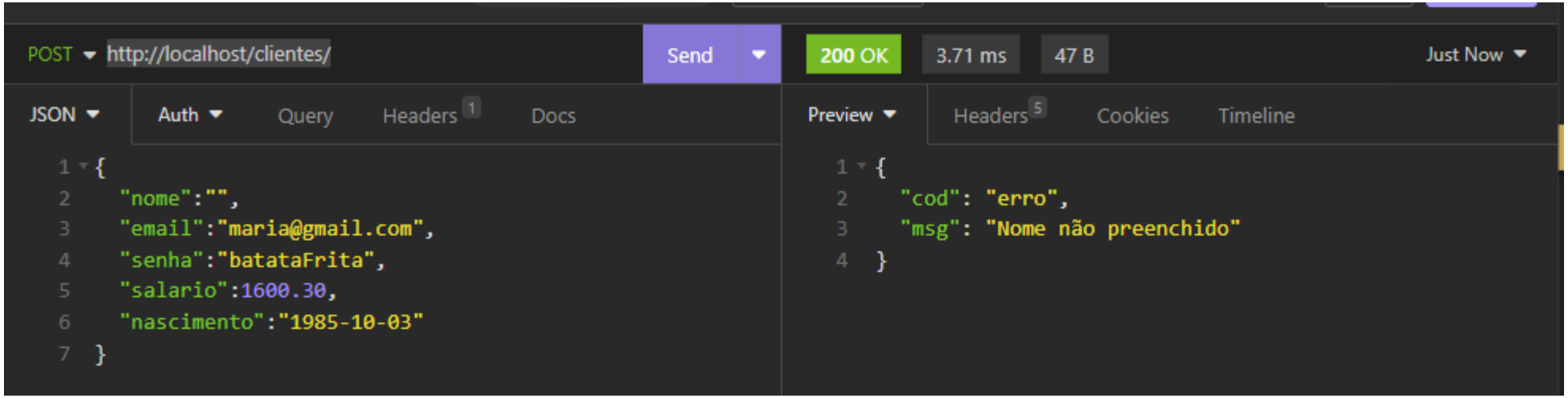
- Resposta: **201 Created**
- Observe que um json é enviado para a rota.
- O recurso retorna uma mensagem do servidor.



Testar o controle http://localhost/clientes/

Resposta: 200 ok

A requisição foi processada, mas não foi realizado o cadastro.



controle/Cliente_atualizar.php -1

```php
<?php
include "modelo/Cliente.php";
$resposta = array();

$jsonRecebido = file_get_contents('php://input');
$obj = json_decode($jsonRecebido);

$nome = strip_tags($obj->nome);
$email = strip_tags($obj->email);
$senha = strip_tags($obj->senha);
$salario = strip_tags($obj->salario);
$nascimento = strip_tags($obj->nascimento);

$partesRota = explode("/", $_SERVER['REQUEST_URI']);
$idCliente = $partesRota[2];
```

Como os dados são enviados dos clientes no corpo da requisição os vetores $_GET e $_POST não podem ser utilizados.
Para recuperar os dados no corpo da requisição utilize:
file_get_contents('php://input');

$obj é um objeto json criado a partir dos dados(texto json) que vieram no corpo da requisição.

PUT	/clientes/id	Controle/Cliente_atualizar.php

Observe que rota na posição 2 é o id do cliente que deve ser atualizado

controle/Cliente_atualizar.php -2

```
//verifica se os dados foram preenchidos corretamente
//este exemplo é simples
//verifique conforme a necessidade da aplicação.
if($nome==""){
    $resposta['cod']="erro";
    $resposta['msg']="Nome não preenchido";

}else if($email==""){
    $resposta['cod']="erro";
    $resposta['msg']="E-mail não preenchido";
}else{
```

controle/Cliente_atualizar.php -3

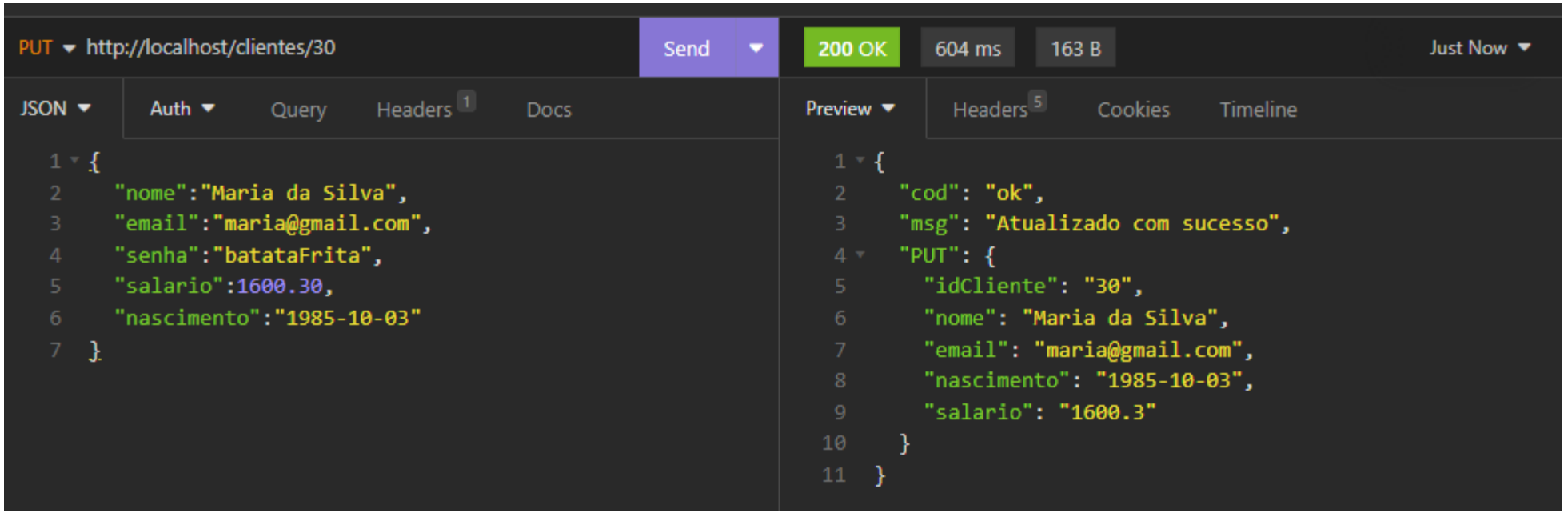
```
$cliente = new Cliente();

 $cliente->setIdCliente($idCliente);
 $cliente->setNome($nome);
 $cliente->setEmail($email);
 $cliente->setSenha($senha);
 $cliente->setSalario($salario);
 $cliente->setNascimento($nascimento);

 $resultado = $cliente->atualizar();
```

controle/Cliente_atualizar.php -4

```
if($resultado==true){
        header("HTTP/1.1 200 OK");
        $resposta['cod']="ok";
        $resposta['msg']="Atualizado com sucesso";
        $resposta['PUT']=$cliente;
    }else{
        header("HTTP/1.1 200 OK");
        $resposta['cod']="erro";
        $resposta['msg']="Não foi atualizado";
    }

}

//converte o array resposta em json para
//enviar de resposta ao cliente
echo json_encode($resposta);
```

Testar o controle
PUT: http://localhost/clientes/id

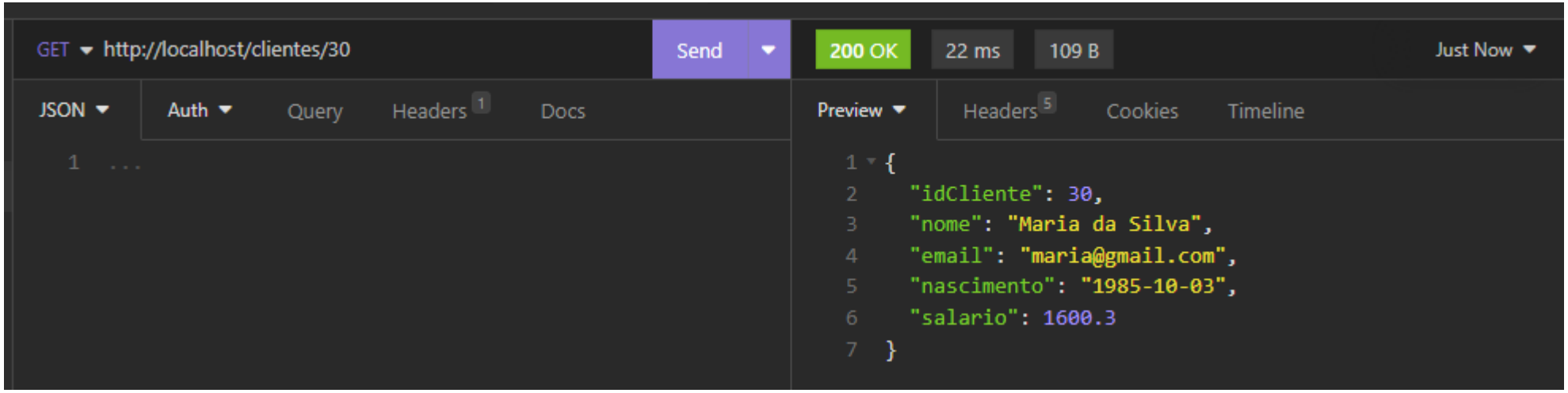
- OBSERVE QUE FOI ENVIADO UM PUT.



controle/Cliente_buscar.php -1

```php
<?php
include "modelo/Cliente.php";
$resposta = array();
$partesRota = explode("/", $_SERVER['REQUEST_URI']);

$id = $partesRota[2];

$cliente = new Cliente();

$clientes = $cliente->buscar($id);

header("HTTP/1.1 200 OK");

echo json_encode($clientes);
```

Testar o controle
GET: http://localhost/clientes/id



controle/Cliente_excluir.php -1

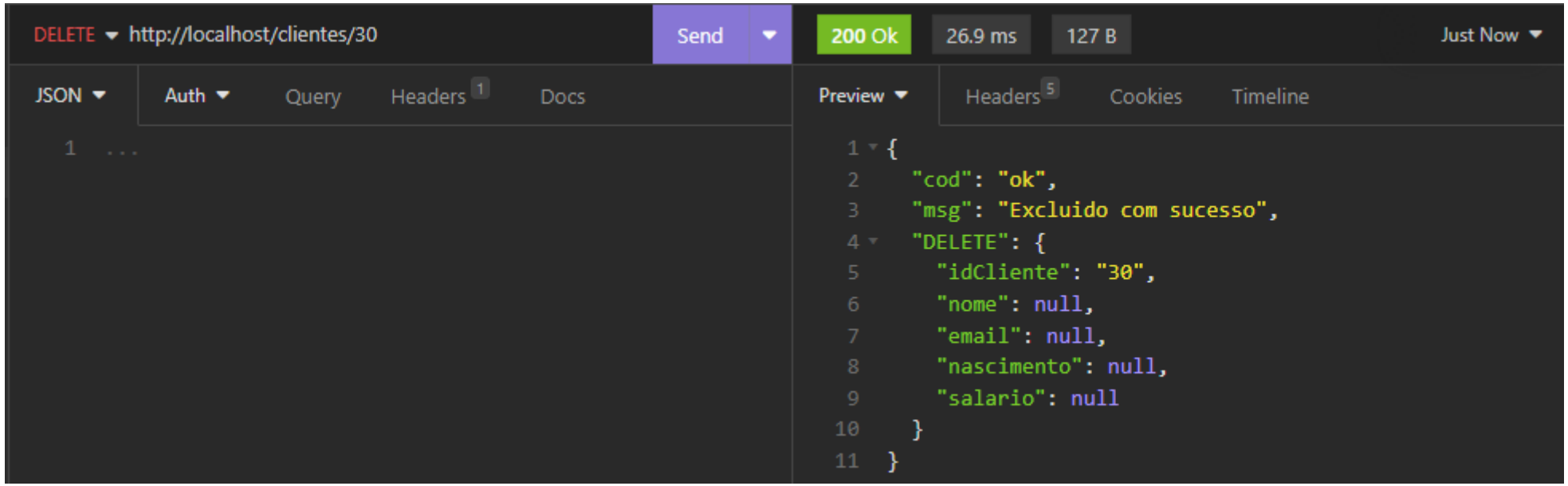
```
include "modelo/Cliente.php";
$resposta = array();
$partesRota = explode("/", $_SERVER['REQUEST_URI']);
$idCliente = $partesRota[2];
$cliente = new Cliente();
$cliente->setIdCliente($idCliente);
$resultado = $cliente->excluir();
```

controle/Cliente_excluir.php -2

```
if ($resultado == true) {
    $resposta['cod'] = "ok";
    $resposta['msg'] = "Excluido com sucesso";
    $resposta['DELETE'] = $cliente;
} else {
    $resposta['cod'] = "erro";
    $resposta['msg'] = "Não foi Excluido";
}

header("HTTP/1.1 200 Ok");
echo json_encode($resposta);
```

Testar o controle
DELETE: http://localhost/clientes/id

- OBSERVE QUE FOI ENVIADO UM DELETE



controle/Cliente_listar.php - 1

```
<?php
include "modelo/Cliente.php";
$resposta = array();

$cliente = new Cliente();

$clientes = $cliente->listar();

header("HTTP/1.1 200 OK");
echo json_encode($clientes);
```

Testar o controle
GET: http://localhost/clientes