REST API

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.

O que é rest

- REST (Representational State Transfer) é uma arquitetura de software utilizada para projetar sistemas de comunicação em rede, geralmente na construção de APIs (*Application Programming Interfaces*).
- é baseada em princípios como a utilização de URIs (*Uniform Resource Identifiers*) para identificar recursos, o uso de métodos HTTP (GET, POST, PUT, DELETE) para operar sobre esses recursos, e a representação dos recursos em formatos como JSON ou XML.
- REST é amplamente adotada na construção de serviços web devido à sua simplicidade, escalabilidade e capacidade de promover a interoperabilidade entre sistemas distribuídos

stateless

- A principal complicação ao criar um serviço REST completamente Stateless surge quando se trata da gestão dos dados de autenticação/autorização dos clientes.
- Isso se torna desafiador porque os desenvolvedores frequentemente têm o hábito de armazenar essas informações em sessões, uma prática comum no desenvolvimento de aplicações Web tradicionais.
- Sessões são sempre armazenadas no servidor... E portanto incompatíveis com a arquitetura rest, pois é armazenado um estado no servidor e a arquitetura define que não são armazenados estados.

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/;
- **Body** : 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/turmas/1b

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 RESTfull

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

Padronização de nomes e métodos:

- Endpoints devem ser compostos unicamente por nomes (substantivos no plural), não use verbos;
- Utilize kebab-case para palavras compostas;
- Use letras minúsculas;
- Não termine seus endpoints com “/”;
- Use diferentes verbos HTTP para suas operações.
  - Por exemplo, POST é usado para criar um Recurso.

Nomes patronizados

GET	http://helioesperidiao.com/produtos;
GET	http://helioesperidiao.com/clientes;
GET	http://helioesperidiao.com/clientes/57;
GET	http://helioesperidiao.com/vendas;

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.

Segurança do sql

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.*

Exemplo cadastrar

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

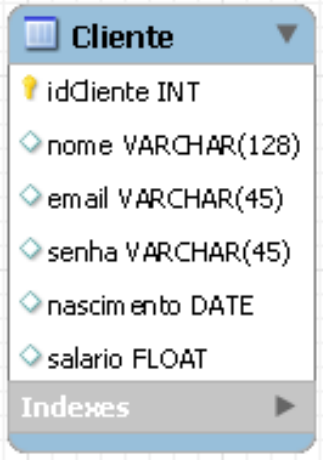
Prepara a instrução sql, posteriormente os "**?**" serão substituídos por valores. Método impede um ataque comum na web chamado de ***sql injection***

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

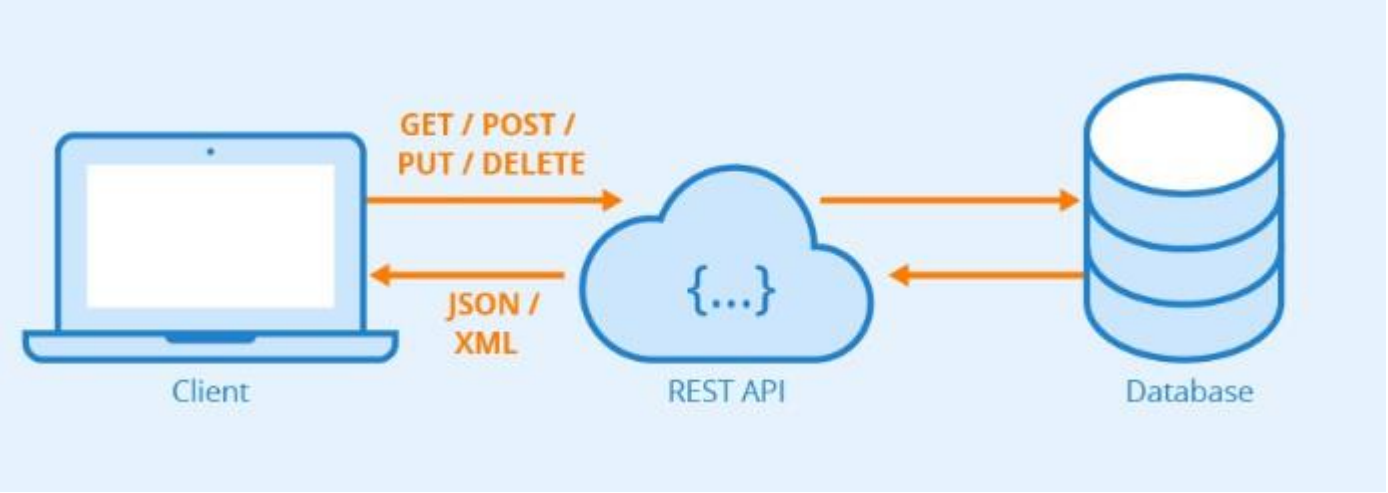
Substitui os ? Pelos valores das variáveis

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

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



Arquitetura de uma aplicação rest



Arquitetura

- .htaccess
- Index.php
- Modelo
  - Cargo.php
  - Funcionario.php
  - Banco.php
- Controle
  - Cargo
    - controle_cargo_create.php
    - controle_cargo_delete.php
    - controle_cargo_read_all.php
    - controle_cargo_read_by_id.php
    - controle_cargo_update.php
  - Funcionario
    - controle_funcionario_create.php
    - controle_funcionario_delete.php
    - controle_funcionario_read_all.php
    - controle_funcionario_read_by_id.php
    - controle_funcionario_update.php

Classes que acessam o banco de dados

- Padronize todos os nomes de campos no banco de dados são atributos da classe.
- Padronize os nomes.
  - Nomes das tabelas:
    - Sempre primeira letra da palavra em maiúscula.
    - Nomes compostos as primeiras letras das palavras maiúsculas:
      - Exemplo: ClienteEspecial
  - Nomes dos campos das tabelas:
    - Sempre a primeira letra da palavra em minúscula.
    - Nomes compostos as primeiras letras das palavras maiúsculas:
      - Exemplo: notaFiscal

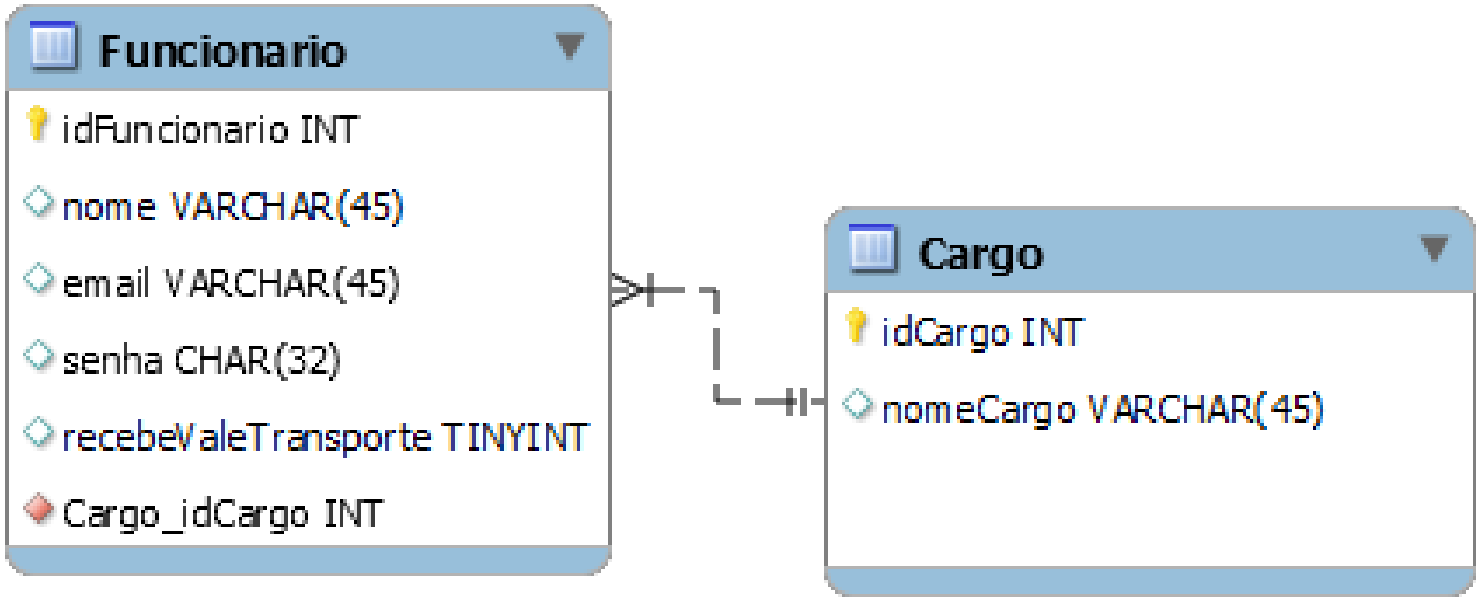
Classes que acessam banco de dados

- Tipicamente essas classes possuem como atributos todos os campos de tabelas do banco de dados.

- Em boa parte dos casos é criada uma classe para cada tabela.

Modelo de banco de dados

- O exemplo construído terá como base o seguinte modelo de banco de dados



Banco.php 1/2

```php
class Banco{
        // Propriedades estáticas para armazenar informações de conexão com o banco de dados
        private static $HOST='127.0.0.1';
        private static $USER= 'root';
        private static $PWD= '';
        private static $DB= 'aula_api_2024';
        private static $PORT= 3306;
        private static $CONEXAO= null;
        // Método privado para estabelecer uma conexão com o banco de dados
        private static function conectar(){
             error_reporting(E_ERROR | E_PARSE);   // Configura o relatório de erros para ocultar mensagens de erro devido a conexões falhas
            // Verifica se já existe uma conexão estabelecida
            if(Banco::$CONEXAO==null){
                // Tenta estabelecer uma nova conexão utilizando as informações fornecidas
                Banco::$CONEXAO = new mysqli(Banco::$HOST,Banco::$USER,Banco::$PWD,Banco::$DB,Banco::$PORT);
                // Verifica se ocorreu algum erro na conexão
                if(Banco::$CONEXAO->connect_error) {
                    // Cria um objeto stdClass para armazenar informações sobre o erro
                    $objResposta = new stdClass();
                    $objResposta->cod = 1;
                    $objResposta->msg = "Erro ao conectar no banco";
                    $objResposta->erro = Banco::$CONEXAO->connect_error;

                    // Encerra o script e retorna o objeto JSON com as informações do erro
                    die(json_encode($objResposta));
                }
            }
        }
```

Banco.php 2/2

```
// Método público para obter a conexão com o banco de dados
        public static function getConexao(){
            // Verifica se já existe uma conexão estabelecida
            if(Banco::$CONEXAO==null){
                // Se não houver, estabelece uma nova conexão
                Banco::conectar();
            }
            // Retorna a conexão
            return Banco::$CONEXAO;
        }
    }
```

Rotas de cargo

Rotas de Cargo

VERBOS	URI(ROTA)	ARQUIVO RESPONSÁVEL
GET	/cargos	controle/cargo/controle_cargo_read_all.php
GET	/cargos/`:idCargo`	controle/cargo/controle_cargo_read_by_id.php
POST	/cargos	controle/cargo/controle_cargo_create.php
PUT	/cargos/`:idCargo`	Controle/cargo/controle_cargo_update.php
DELETE	/cargos/`:idCargo`	controle/cargo/controle_cargo_delete.php

```php
<?php
// Inclui o arquivo Router.php, que provavelmente contém a definição da classe Router
require_once ("modelo/Router.php");
// Instancia um objeto da classe Router
$roteador = new Router();

// Define uma rota para a obtenção de todos os cargos
$roteador->get("/cargos", function () {
    // Requer o arquivo de controle responsável por obter todos os cargos
    require_once ("controle/cargo/controle_cargo_read_all.php");
});

// Define uma rota para a obtenção de um cargo específico pelo ID
$roteador->get("/cargos/(\d+)", function ($idCargo) {
    // Requer o arquivo de controle responsável por obter um cargo pelo ID
    require_once ("controle/cargo/controle_cargo_read_by_id.php");
});

// Define uma rota para a criação de um novo cargo
$roteador->post("/cargos", function () {
    // Requer o arquivo de controle responsável por criar um novo cargo
    require_once ("controle/cargo/controle_cargo_create.php");
});

// Define uma rota para a atualização de um cargo existente pelo ID
$roteador->put("/cargos/(\d+)", function ($idCargo) {
    // Requer o arquivo de controle responsável por atualizar um cargo pelo ID
    require_once ("controle/cargo/controle_cargo_update.php");
});
// Define uma rota para a exclusão de um cargo existente pelo ID
$roteador->delete("/cargos/(\d+)", function ($idCargo) {
    // Requer o arquivo de controle responsável por excluir um cargo pelo ID
    require_once ("controle/cargo/controle_cargo_delete.php");
});
```

Cargo.php

```php
// Inclui o arquivo Banco.php, que provavelmente contém funcionalidades relacionadas ao banco de dados
require_once ("modelo/Banco.php");

// Definição da classe Cargo, que implementa a interface JsonSerializable
class Cargo implements JsonSerializable{
    // Propriedades privadas da classe
    private $idCargo;
    private $nomeCargo;
```

Cargo.php

```
//se um objeto da classe cargo for passado como paramentro para a função echo json_encode($ojetoCargo);
//o método jsonSerialize() é chamado
// Método necessário pela interface JsonSerializable para serialização do objeto para JSON
    public function jsonSerialize(){
        // Cria um objeto stdClass para armazenar os dados do cargo
        $objetoResposta = new stdClass();
        // Define as propriedades do objeto com os valores das propriedades da classe
        $objetoResposta->idCargo = $this->idCargo;
        $objetoResposta->nomeCargo = $this->nomeCargo;

        // Retorna o objeto para serialização json_encode($ojetoCargo)
        return $objetoResposta;
    }
```

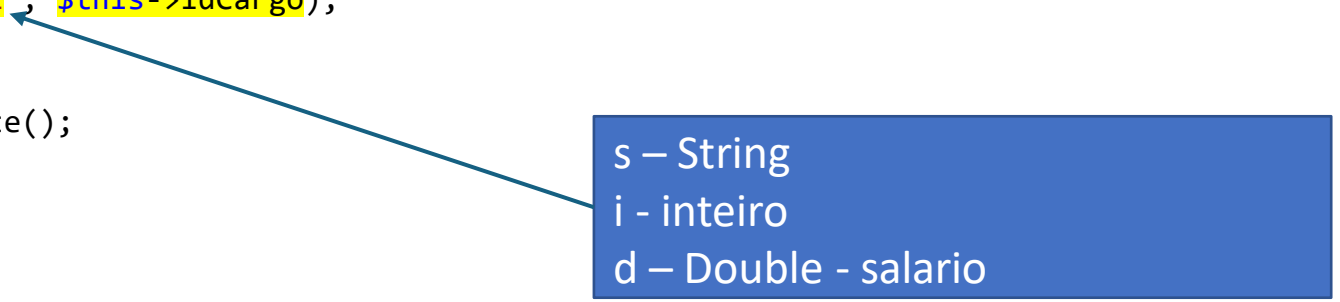
Cargo.php

```php
// Método para criar um novo cargo no banco de dados
    public function create(){
        // Obtém a conexão com o banco de dados
        $conexao = Banco::getConexao();
        // Define a consulta SQL para inserir um novo cargo
        $SQL = "INSERT INTO cargo (nomeCargo)VALUES(?);";
        // Prepara a consulta
        $prepareSQL = $conexao->prepare($SQL);
        // Define o parâmetro da consulta com o nome do cargo
        $prepareSQL->bind_param("s", $this->nomeCargo);
        // Executa a consulta
        $executou = $prepareSQL->execute();
        // Obtém o ID do cargo inserido
        $idCadastrado = $conexao->insert_id;
        // Define o ID do cargo na instância atual da classe
        $this->setIdCargo($idCadastrado);
        // Retorna se a operação foi executada com sucesso
        return $executou;
    }
```

s – String
i - inteiro
d – Double - salario

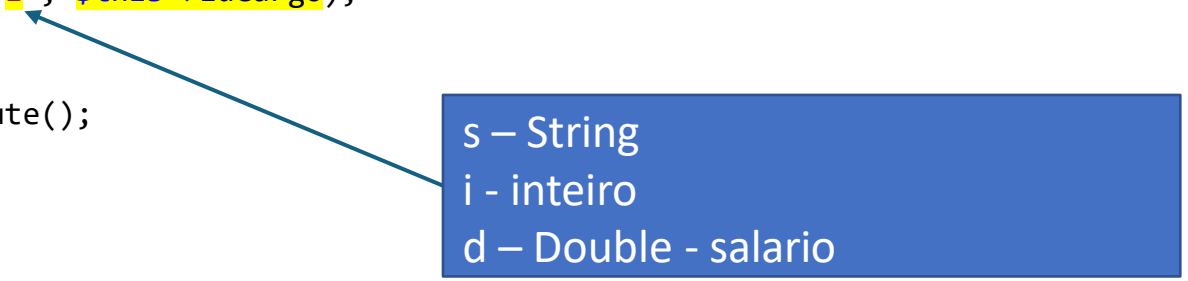
Cargo.php

```php
// Método para excluir um cargo do banco de dados
    public function delete(){
        // Obtém a conexão com o banco de dados
        $conexao = Banco::getConexao();
        // Define a consulta SQL para excluir um cargo pelo ID
        $SQL = "delete from cargo where idCargo=?;";
        // Prepara a consulta
        $prepareSQL = $conexao->prepare($SQL);
        // Define o parâmetro da consulta com o ID do cargo
        $prepareSQL->bind_param("i", $this->idCargo);
        // Executa a consulta
        return $prepareSQL->execute();
    }
```



Cargo.php

```php
// Método para excluir um cargo do banco de dados
public function delete(){
    // Obtém a conexão com o banco de dados
    $conexao = Banco::getConexao();
    // Define a consulta SQL para excluir um cargo pelo ID
    $SQL = "delete from cargo where idCargo=?;";
    // Prepara a consulta
    $prepareSQL = $conexao->prepare($SQL);
    // Define o parâmetro da consulta com o ID do cargo
    $prepareSQL->bind_param("i", $this->idCargo);
    // Executa a consulta
    return $prepareSQL->execute();
}
```



Cargo.php

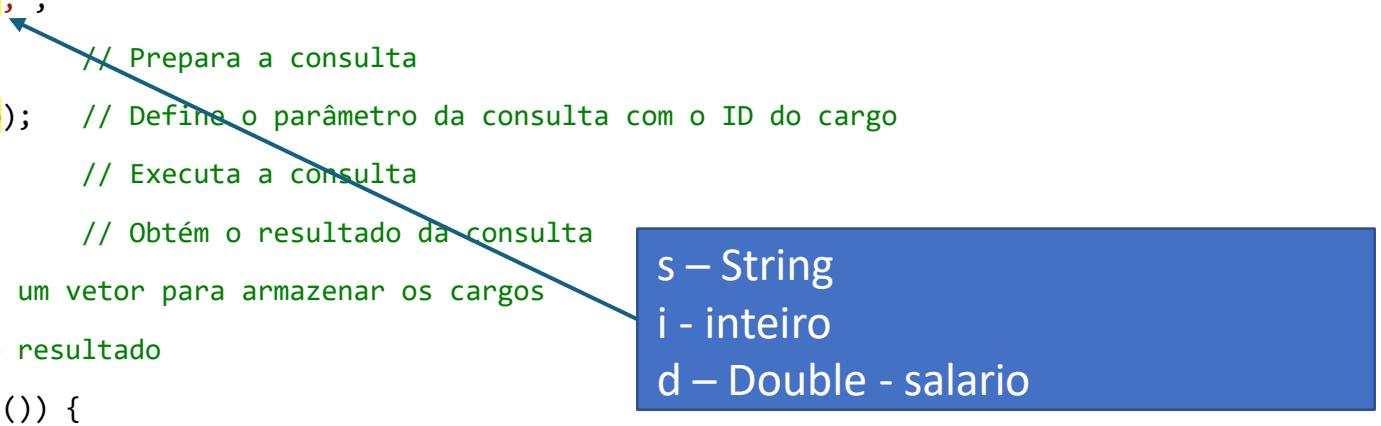
```php
// Método para verificar se um cargo já existe no banco de dados
   public function isCargo()     {
        // Obtém a conexão com o banco de dados
        $conexao = Banco::getConexao();
        // Define a consulta SQL para contar quantos cargos possuem o mesmo nome
        $SQL = "SELECT COUNT(*) AS qtd FROM cargo WHERE nomeCargo =?;";
        // Prepara a consulta
        $prepareSQL = $conexao->prepare($SQL);
        // Define o parâmetro da consulta com o nome do cargo
        $prepareSQL->bind_param("s", $this->nomeCargo);
        // Executa a consulta
        $executou = $prepareSQL->execute();
        // Obtém o resultado da consulta
        $matrizTuplas = $prepareSQL->get_result();
        // Extrai o objeto da tupla
        $objTupla = $matrizTuplas->fetch_object();
        // Retorna se a quantidade de cargos encontrados é maior que zero – true || false
        return $objTupla->qtd > 0;

   }
```

Cargo.php

```php
// Método para ler todos os cargos do banco de dados
    public function readAll(){
        $conexao = Banco::getConexao(); // Obtém a conexão com o banco de dados
        $SQL = "Select * from cargo order by nomeCargo";
        // Prepara a consulta
        $prepareSQL = $conexao->prepare($SQL);
        $executou = $prepareSQL->execute(); // Executa a consulta
        $matrizTuplas = $prepareSQL->get_result(); // Obtém o resultado da consulta
        $vetorCargos = array();   // Inicializa um vetor para armazenar os cargos
        $i = 0; // Itera sobre as tuplas do resultado
        while ($tupla = $matrizTuplas->fetch_object()) {
            $vetorCargos[$i] = new Cargo(); // Cria uma nova instância de Cargo para cada tupla encontrada
            // Define o ID e o nome do cargo na instância
            $vetorCargos[$i]->setIdCargo($tupla->idCargo);
            $vetorCargos[$i]->setNomeCargo($tupla->nomeCargo);
            $i++;
        }
        // Retorna o vetor com os cargos encontrados
        return $vetorCargos;
    }
```

Cargo.php

```php
// Método para ler um cargo do banco de dados com base no ID
    public function readByID() {
        $conexao = Banco::getConexao(); // Obtém a conexão com o banco de dados
        $SQL = "SELECT * FROM cargo WHERE idCargo=?;";
        $prepareSQL = $conexao->prepare($SQL);          // Prepara a consulta
        $prepareSQL->bind_param("i", $this->idCargo);   // Define o parâmetro da consulta com o ID do cargo
        $executou = $prepareSQL->execute();             // Executa a consulta
        $matrizTuplas = $prepareSQL->get_result();      // Obtém o resultado da consulta
        $vetorCargos = array();       // Inicializa um vetor para armazenar os cargos
        $i = 0;         // Itera sobre as tuplas do resultado
        while ($tupla = $matrizTuplas->fetch_object()) {
            // Cria uma nova instância de Cargo para cada tupla encontrada
            $vetorCargos[$i] = new Cargo();
            // Define o ID e o nome do cargo na instância
            $vetorCargos[$i]->setIdCargo($tupla->idCargo);
            $vetorCargos[$i]->setNomeCargo($tupla->nomeCargo);
            $i++;
        }
        // Retorna o vetor com os cargos encontrados
        return $vetorCargos;
    }
```



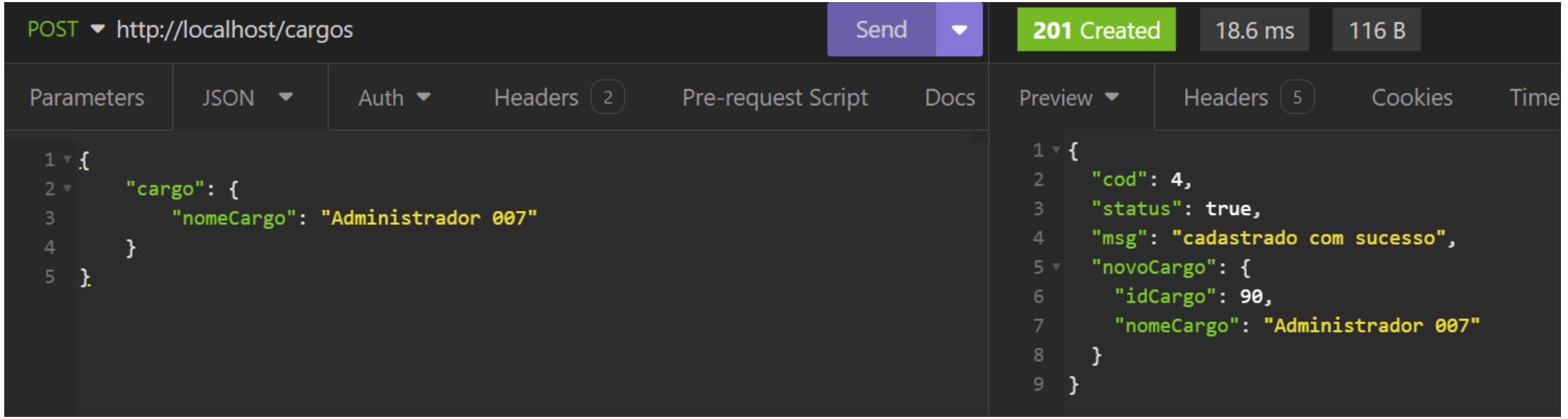
Cargo.php

```php
// Método getter para idCargo
    public function getIdCargo()
    {
        return $this->idCargo;
    }

    // Método setter para idCargo
    public function setIdCargo($idCargo)
    {
        $this->idCargo = $idCargo;

        return $this;
    }

    // Método getter para nomeCargo
    public function getNomeCargo()
    {
        return $this->nomeCargo;
    }

    // Método setter para nomeCargo
    public function setNomeCargo($nameCargo)
    {
        $this->nomeCargo = $nameCargo;

        return $this;
    }
}
```

controle_cargo_create.php

Envio json e resposta esperada

- Seguindo o padrão rest para criar um novo objeto é necessário enviar um post com os dados do objeto para a rota.



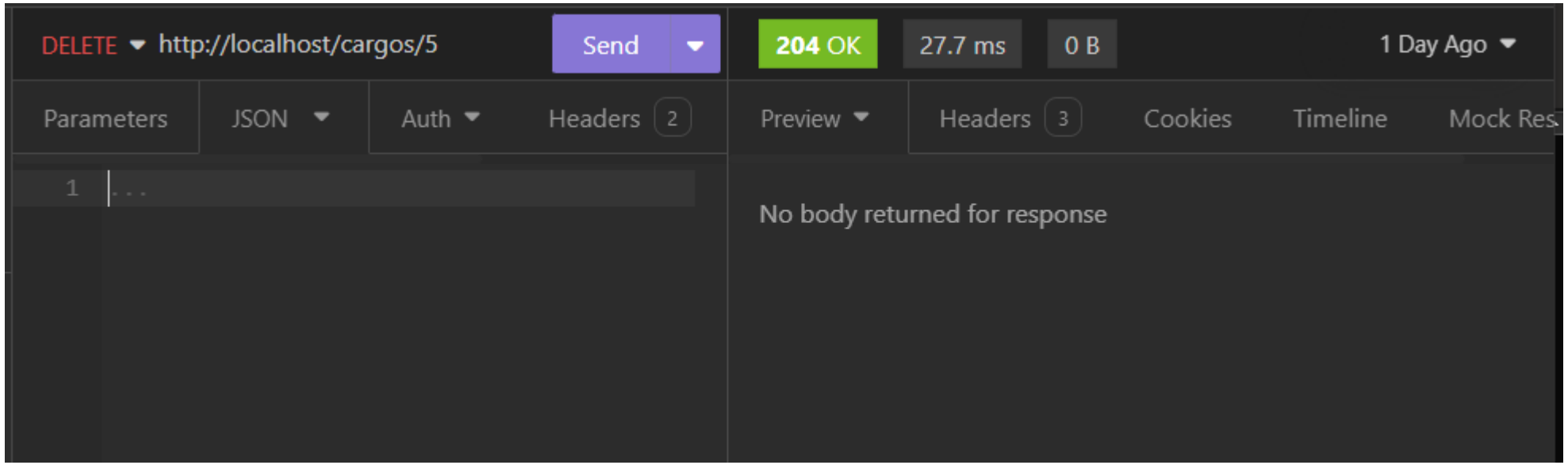
```php
// Inclui as classes Banco e Cargo, que contêm funcionalidades relacionadas ao banco de dados e aos cargos
require_once ("modelo/Banco.php");
require_once ("modelo/Cargo.php");

// Obtém os dados enviados por meio de uma requisição POST em formato JSON
$textoRecebido = file_get_contents("php://input");
// Decodifica os dados JSON recebidos em um objeto PHP ou interrompe o script se o formato estiver incorreto
$objJson = json_decode($textoRecebido) or die('{"msg":"formato incorreto"}');

// Cria um novo objeto para armazenar a resposta
$objResposta = new stdClass();
// Cria um novo objeto da classe Cargo
$objCargo = new Cargo();

// Define o nome do cargo recebido do JSON no objeto Cargo
$objCargo->setNomeCargo($objJson->cargo->nomeCargo);
```

```php
if ($objCargo->getNomeCargo() == "") {// Verifica se o nome do cargo está vazio
    $objResposta->cod = 1;
    $objResposta->status = false;
    $objResposta->msg = "o nome nao pode ser vazio";
} else if (strlen($objCargo->getNomeCargo()) < 3) {// Verifica se o nome do cargo tem menos de 3 caracteres
    $objResposta->cod = 2;
    $objResposta->status = false;
    $objResposta->msg = "o nome nao pode ser menor do que 3 caracteres";
} else if ($objCargo->isCargo() == true) {// Verifica se já existe um cargo cadastrado com o mesmo nome
    $objResposta->cod = 3;
    $objResposta->status = false;
    $objResposta->msg = "Ja existe um cargo cadastrado com o nome: " . $objCargo->getNomeCargo();
} else {// Se todas as condições anteriores forem atendidas, tenta criar um novo cargo
    if ($objCargo->create() == true) {    // Verifica se a criação do novo cargo foi bem-sucedida
        $objResposta->cod = 4;
        $objResposta->status = true;
        $objResposta->msg = "cadastrado com sucesso";
        $objResposta->novoCargo = $objCargo;
    }else {    // Se houver erro na criação do cargo, define a mensagem de erro
        $objResposta->cod = 5;
        $objResposta->status = false;
        $objResposta->msg = "Erro ao cadastrar novo Cargo";
    }
}
```

```php
// Define o tipo de conteúdo da resposta como JSON
header("Content-Type: application/json");

// Define o código de status da resposta com base no status da operação
if ($objResposta->status == true) {
    header("HTTP/1.1 201");
} else {
    header("HTTP/1.1 200");
}

// Converte o objeto resposta em JSON e o imprime na saída
echo json_encode($objResposta);
```

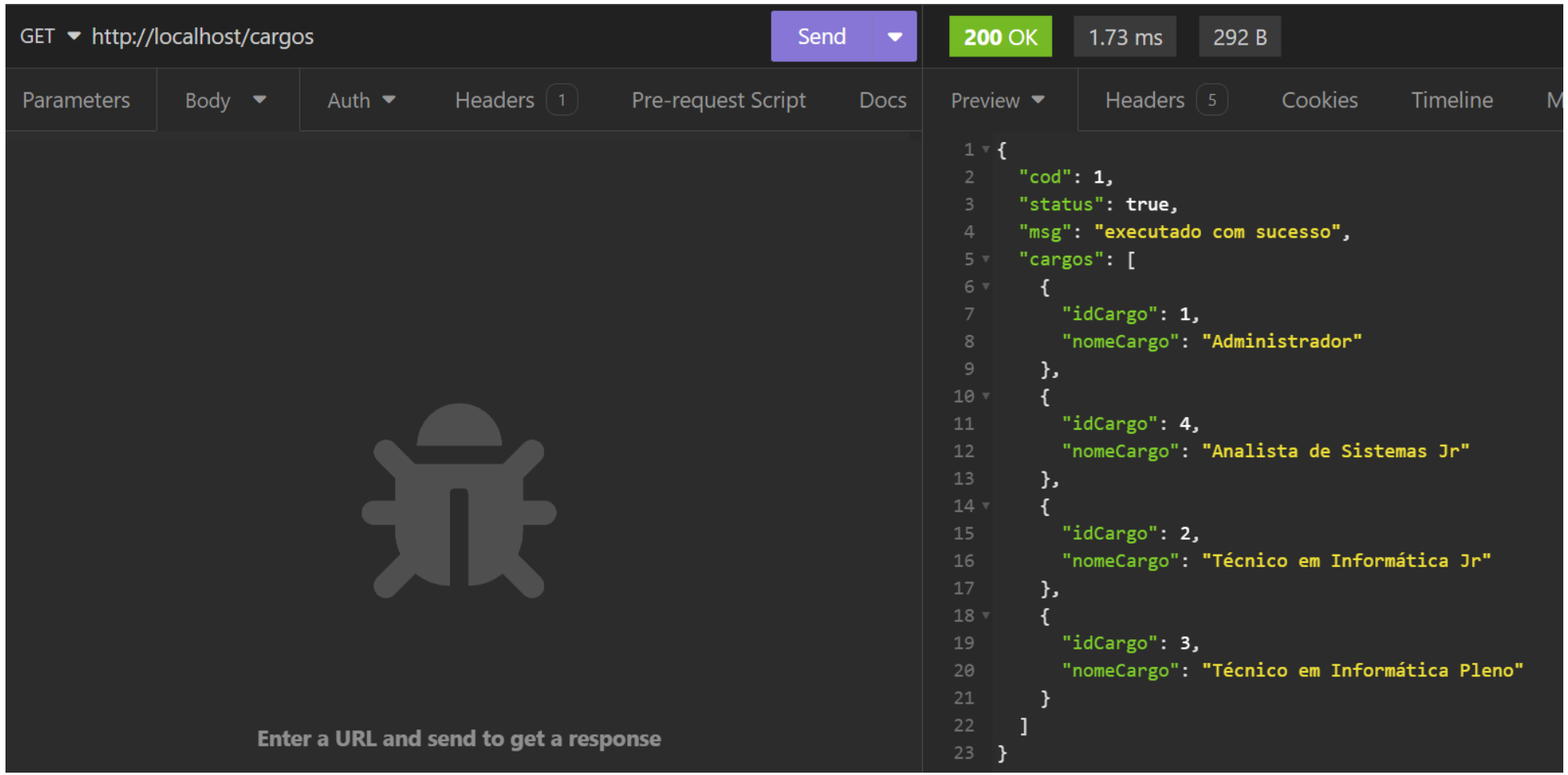
controle_cargo_delete.php

- Observe que não é enviado um json
- Observe que é solicitada a exclusão do cargo de id igual a cinco.
- Observe que segundo a especificação do código http 204 não há corpo de resposta



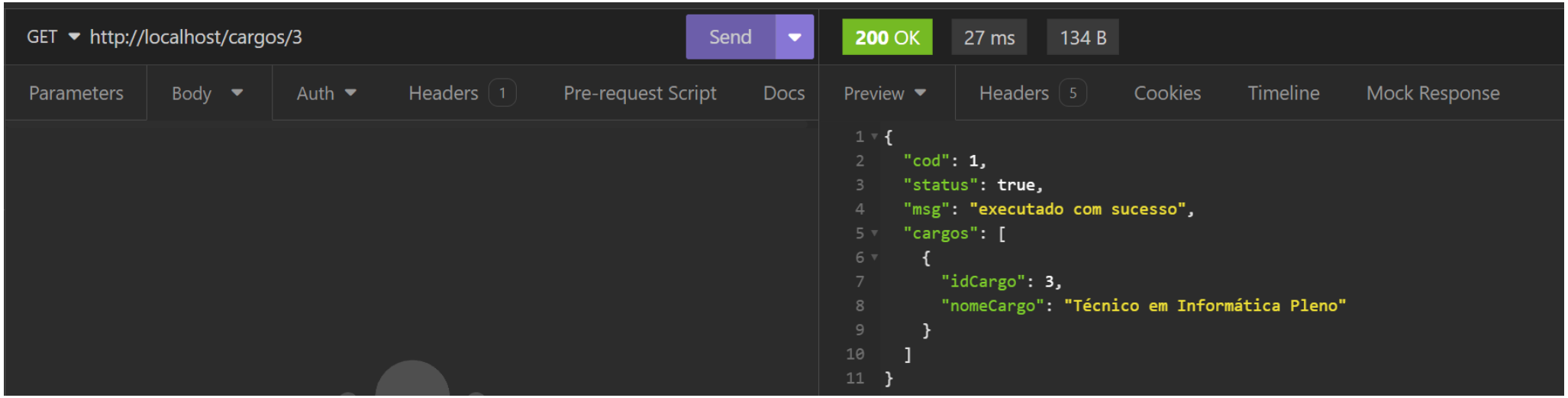
```php
<?php
require_once ("modelo/Cargo.php"); // Inclui a classe Cargo.php, que provavelmente contém funcionalidades relacionadas aos cargos
$objResposta = new stdClass(); // Cria um novo objeto para armazenar a resposta
$objCargo = new Cargo(); // Cria um novo objeto da classe Cargo
$objCargo->setIdCargo($idCargo); // Define o ID do cargo a ser excluído
if($objCargo->delete()==true){// Verifica se a exclusão do cargo foi bem-sucedida
    header("HTTP/1.1 204");     // Define o código de status da resposta como 204 (No Content)
}else{
    header("HTTP/1.1 200");     // Define o código de status da resposta como 200 (OK)
    header("Content-Type: application/json");     // Define o tipo de conteúdo da resposta como JSON
    $objResposta->status = false;        // Define o status da resposta como falso
    $objResposta->cod = 1;     // Define o código de resposta como 1
    // Define a mensagem de erro
    $objResposta->msg = "Erro ao excluir cargo";
    // Converte o objeto resposta em JSON e o imprime na saída
    echo json_encode($objResposta);
}
?>
```

controle_cargo_read_all.php


GET http://localhost/cargos
Send
200 OK
1.73 ms
292 B
Parameters
Body
Auth
Headers 1
Pre-request Script
Docs
Preview
Headers 5
Cookies
Timeline
Enter a URL and send to get a response
{
"cod": 1,
"status": true,
"msg": "executado com sucesso",
"cargos": [
{
"idCargo": 1,
"nomeCargo": "Administrador"
},
{
"idCargo": 4,
"nomeCargo": "Analista de Sistemas Jr"
},
{
"idCargo": 2,
"nomeCargo": "Técnico em Informática Jr"
},
{
"idCargo": 3,
"nomeCargo": "Técnico em Informática Pleno"
}
]
}

```php
<?php

// Inclui as classes Banco e Cargo, que contêm funcionalidades relacionadas ao banco de dados e aos cargos

require_once ("modelo/Banco.php");

require_once ("modelo/Cargo.php");
$objResposta = new stdClass(); // Cria um novo objeto para armazenar a resposta

$objCargo = new Cargo(); // Cria um novo objeto da classe Cargo
// Obtém todos os cargos do banco de dados

$vetor = $objCargo ->readAll(); // vetor de objetos do tipo Cargo

$objResposta->cod = 1;
// Define o código de resposta como 1

// Define o status da resposta como verdadeiro

$objResposta->status = true;

// Define a mensagem de sucesso

$objResposta->msg = "executado com sucesso";

// Define o vetor de cargos na resposta

$objResposta->cargos = $vetor;
// Define o código de status da resposta como 200 (OK)

header("HTTP/1.1 200");

// Define o tipo de conteúdo da resposta como JSON

header("Content-Type: application/json");

// Converte o objeto resposta em JSON e o imprime na saída

echo json_encode($objResposta);


?>
```

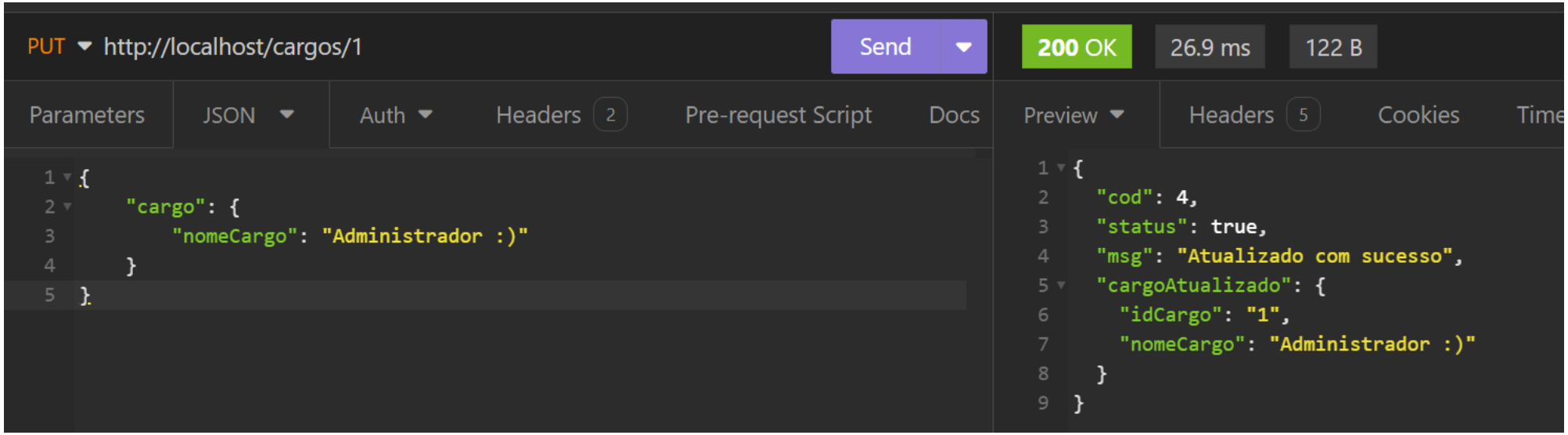
controle_cargo_read_by_id.php


GET http://localhost/cargos/3
Send
200 OK
27 ms
134 B
Parameters
Body
Auth
Headers 1
Pre-request Script
Docs
Preview
Headers 5
Cookies
Timeline
Mock Response
{
"cod": 1,
"status": true,
"msg": "executado com sucesso",
"cargos": [
{
"idCargo": 3,
"nomeCargo": "Técnico em Informática Pleno"
}
]
}

```php
<?php
// Inclui as classes Banco e Cargo, que contêm funcionalidades relacionadas ao banco de dados e aos cargos
require_once ("modelo/Banco.php");
require_once ("modelo/Cargo.php");
// Cria um novo objeto para armazenar a resposta
$objResposta = new stdClass();
// Cria um novo objeto da classe Cargo
$objCargo = new Cargo();
// Define o ID do cargo a ser lido
$objCargo->setIdCargo($idCargo);
$vetor = $objCargo ->readByID(); // Obtém o cargo específico do banco de dados com base no ID fornecido
$objResposta->cod = 1; // Define o código de resposta como 1
$objResposta->status = true; // Define o status da resposta como verdadeiro
$objResposta->msg = "executado com sucesso"; // Define a mensagem de sucesso
$objResposta->cargos = $vetor; // Define o vetor de cargos na resposta
header("HTTP/1.1 200"); // Define o código de status da resposta como 200 (OK)
header("Content-Type: application/json"); // Define o tipo de conteúdo da resposta como JSON
// Converte o objeto resposta em JSON e o imprime na saída
echo json_encode($objResposta);


?>
```

controle_cargo_update.php

- Observe que é enviado o novo nome de cargo para o cargo de id igual a um.
- Observe que há uma resposta json.



```php
<?php
// Inclui as classes Banco e Cargo, que contêm funcionalidades relacionadas ao banco de dados e aos cargos
require_once ("modelo/Banco.php");
require_once ("modelo/Cargo.php");

// Obtém os dados enviados por meio de uma requisição POST em formato JSON
$textoRecebido = file_get_contents("php://input");
// Decodifica os dados JSON recebidos em um objeto PHP ou interrompe o script se o formato estiver incorreto
$objJson = json_decode($textoRecebido) or die('{"msg":"formato incorreto"}');

// Cria um novo objeto para armazenar a resposta
$objResposta = new stdClass();
// Cria um novo objeto da classe Cargo
$objCargo = new Cargo();
// Define o ID do cargo a ser atualizado
$objCargo->setIdCargo($idCargo);
// Define o nome do cargo com base nos dados recebidos do JSON
$objCargo->setNomeCargo($objJson->cargo->nomeCargo);
```

```php
if ($objCargo->getNomeCargo() == "") {// Verifica se o nome do cargo está vazio
    $objResposta->cod = 1;
    $objResposta->status = false;
    $objResposta->msg = "o nome nao pode ser vazio";
}  else if (strlen($objCargo->getNomeCargo()) < 3) {// Verifica se o nome do cargo tem menos de 3 caracteres
    $objResposta->cod = 2;
    $objResposta->status = false;
    $objResposta->msg = "o nome nao pode ser menor do que 3 caracteres";
} else if ($objCargo->isCargo() == true) {// Verifica se já existe um cargo cadastrado com o mesmo nome
    $objResposta->cod = 3;
    $objResposta->status = false;
    $objResposta->msg = "Ja existe um cargo cadastrado com o nome: " . $objCargo->getNomeCargo();
} else {// Se todas as condições anteriores forem atendidas, tenta atualizar o cargo
        if ($objCargo->update() == true) {// Verifica se a atualização do cargo foi bem-sucedida
        $objResposta->cod = 4;
        $objResposta->status = true;
        $objResposta->msg = "Atualizado com sucesso";
        $objResposta->cargoAtualizado = $objCargo;
    }else {// Se houver erro na atualização do cargo, define a mensagem de erro
        $objResposta->cod = 5;
        $objResposta->status = false;
        $objResposta->msg = "Erro ao cadastrar novo Cargo";
    }
}
```

```php
// Define o código de status da resposta como 200 (OK)
header("HTTP/1.1 200");
// Define o tipo de conteúdo da resposta como JSON
header("Content-Type: application/json");
// Converte o objeto resposta em JSON e o imprime na saída
echo json_encode($objResposta);
```

Rotas funcionários

Rotas de Cargo

VERBOS	URI(ROTA)	ARQUIVO RESPONSÁVEL
GET	/funcionários	controle/funcionario/controle_funcionario_read_all.php
GET	/funcionarios/`idFuncionario`	controle/funcionario/controle_funcionario_read_by_id.php
POST	/funcionarios	controle/funcionario/controle_funcionario_create.php
PUT	/funcionarios/`idFuncionario`	Controle/funcionario/controle_funcionario_update.php
DELETE	/funcionarios/`idFuncionario`	controle/funcionario/controle_funcionario_delete.php

- Acompanhe a codificação da classe funcionário nos arquivos de aula.