

REST API – [Cargo/Funcionario] em php 8

Prof. Me. Hélio Esperidião





Organização do sistema.

Regras, regras e mais regras!!!

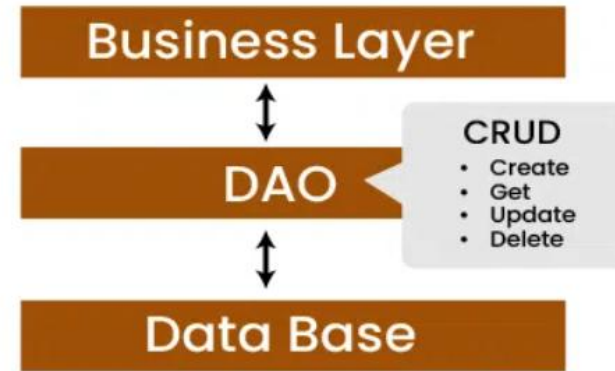
Tecnologia utilizada

- PHP 8
- REST API
- POO
- DAO



Data Access Object

Design Pattern



DAO

DAO:

Data Access Object (Objeto de Acesso a Dados).

- É um padrão de projeto (**design pattern**).
- DAO é Usado para separar a lógica de acesso a dados da lógica de negócios.
 - Lógica de Negócios
 - Regras do sistema, o que deve acontecer com os dados
 - Se o salário for maior que R\$ 5.000, aplicar um bônus de 10%
 - Lógica de Acesso a Dados
 - Como os dados são lidos ou gravados no banco de dados.Exemplo: “
 - Buscar o funcionário no banco pelo ID”: “Select * from funcionário Where id=1”
- DAO lida Apenas com a lógica de acesso a dados
- Facilita manutenção e testes.



Por que usar DAO?

- Isolamento do código de acesso ao banco de dados.
 - Facilita mudanças no banco sem afetar outras partes do sistema.
 - Reutilização de código.
 - Aumenta a organização do projeto.
-

Estrutura Básica de um DAO

```
<?php

require_once "api/model/Cargo.php";

class TabelaDAO{

    public function create(Tabela $objeto):Tabela |false { /*...*/ }
    public function readAll(): array{ /*...*/ }
    public function readByPage(int $page, int $limit): array{ /*...*/ }
    public function readById(int $id):Tabela { /*...*/ }
    public function readByName(string $nomeCargo):Tabela |null { /*...*/ }
    public function update(Tabela $objeto):bool { /*...*/ }
    public function delete(Tabela $objeto):bool { /*...*/ }

}
```

Modelos em DAO

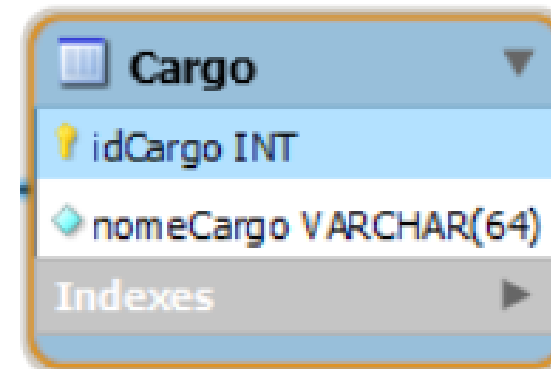


Modelo em um DAO

- O modelo é uma classe que representa por meio de seus atributos uma tabela do banco de dados e possui métodos de get e set
- Padronize todos os nomes de campos no banco de dados, eles serão 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

Diagrama de classes de um Modelo

 Cargo
- idCargo: int - nomeCargo: string
+ __construct(idCargo:int, nomeCargo:string) + jsonSerialize():array + getIdCargo(): int null + setIdCargo(nomeCargo: string): selft + getNomeCargo(): string + seNomeCargo(nomeCargo: string): selft



```

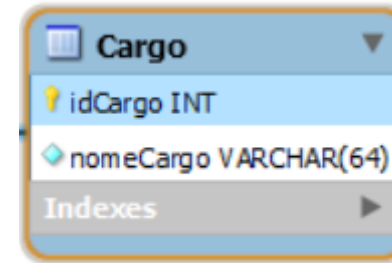
<?php
declare(strict_types=1);
class Cargo implements JsonSerializable {
    public function __construct(
        private ?int $idCargo = null,
        private string $nomeCargo = "",
    ) {}

    public function jsonSerialize(): array{
        return [
            'idCargo' => $this->idCargo,
            'nomeCargo' => $this->nomeCargo
        ];
    }
    public function getIdCargo(): int|null {
        return $this->idCargo;
    }
    public function setIdCargo(int $idCargo): self {

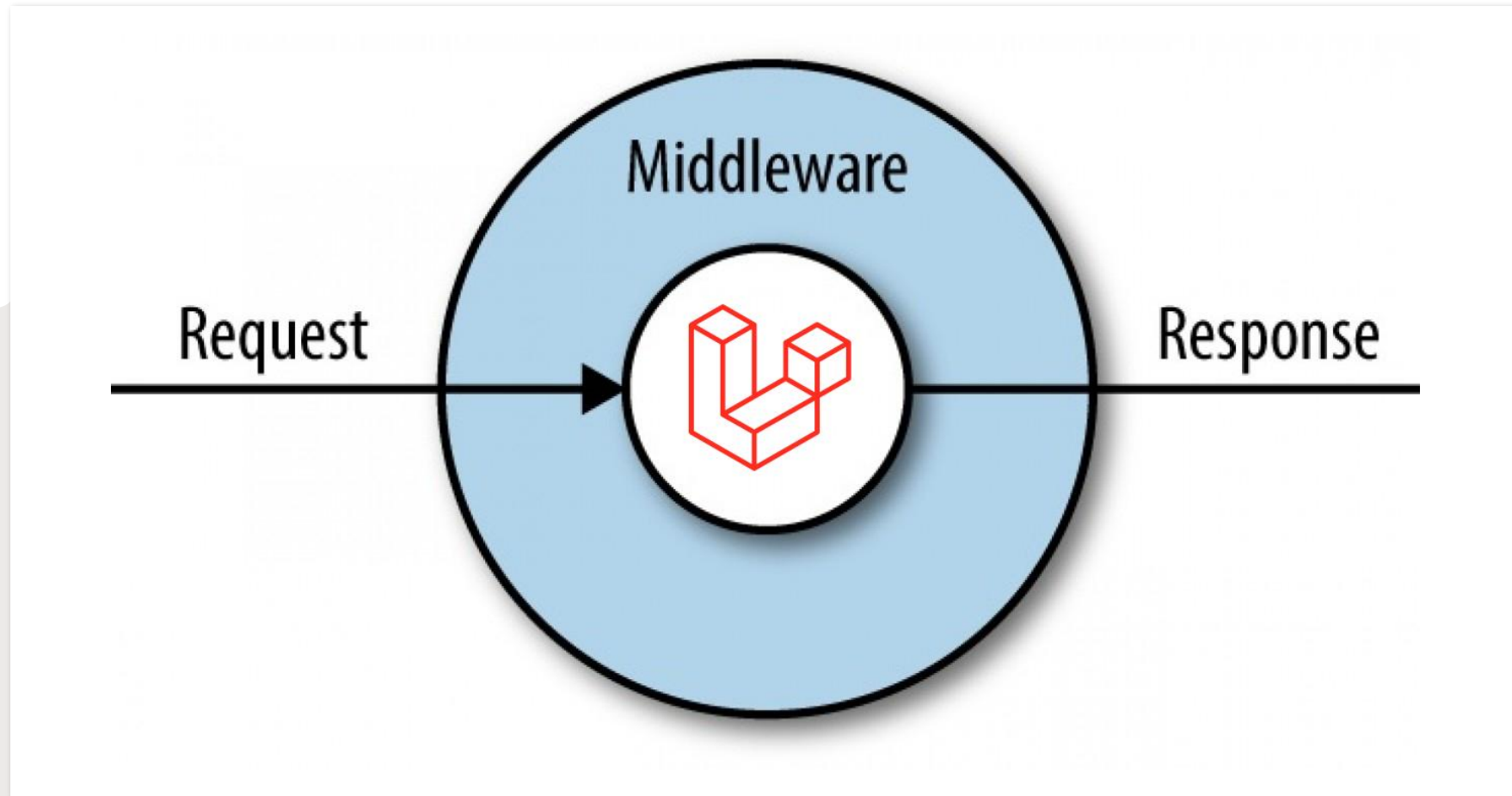
        $this->idCargo = $idCargo;
        return $this;
    }
    public function getNomeCargo(): string {
        return $this->nomeCargo;
    }
    public function setNomeCargo(string $nomeCargo): self{
        $this->nomeCargo = $nomeCargo;
        return $this;
    }
}

```

Modelo



Cargo
- idCargo: int - nomeCargo: string
+ __construct(idCargo:int, nomeCargo:string) + jsonSerialize():array + getIdCargo(): int null + setIdCargo(nomeCargo: string): selft + getNomeCargo(): string + seNomeCargo(nomeCargo: string): selft



Middleware

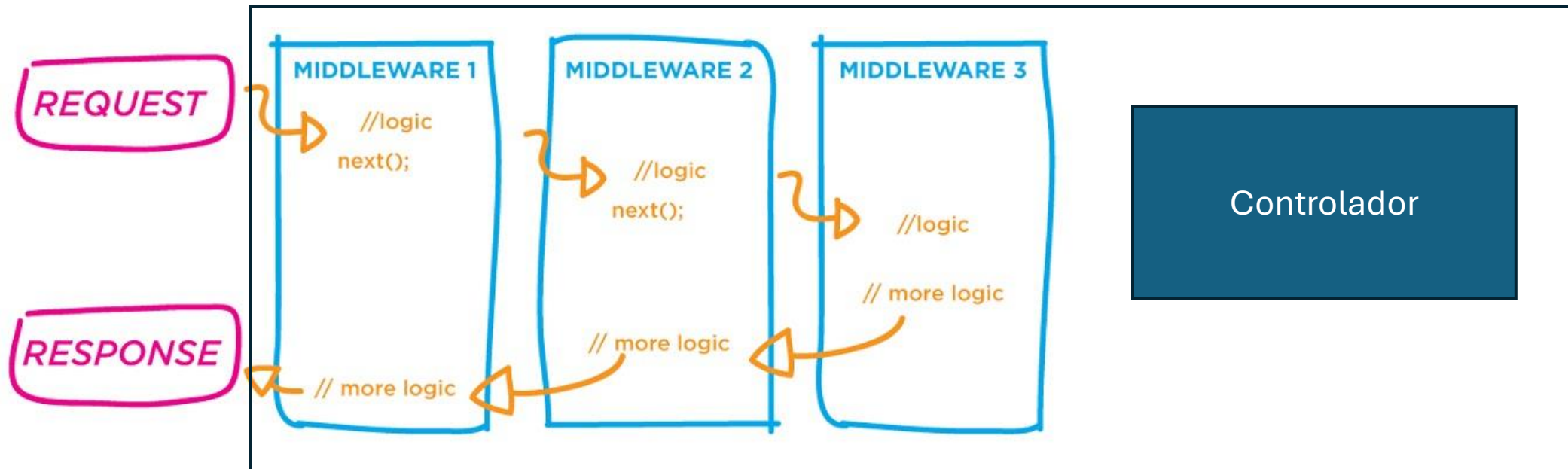


O que é Middleware?

- Função intermediária entre requisição e resposta.
 - Pode atuar antes ou depois do roteamento.
 - Ideal para validações, autenticação, logs, tratamento de erros.
-

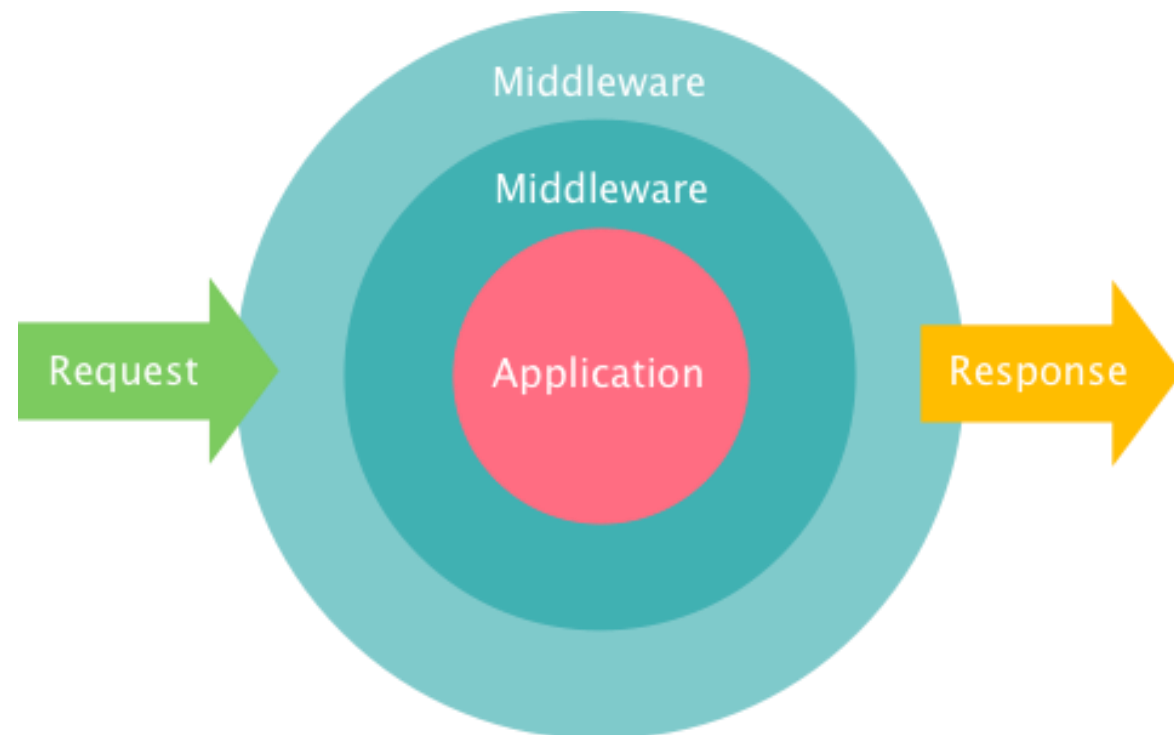
Middleware Pós-Roteamento

- Executado após o roteamento identificar o endpoint
- Utilizado para verificar se os dados recebidos estão corretos.
- Evita que dados inválidos cheguem aos **CONTROLADORES**.





A Aplicação real é a manipulação
do banco de dados
os middlewares de validação são
camadas de segurança até o
acesso ao banco



Para que serve um Middleware?

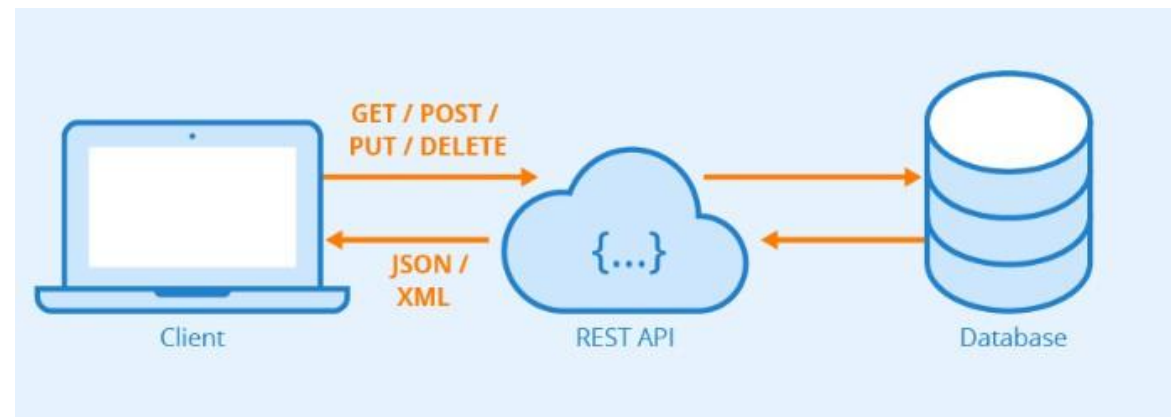
- Autenticação e Autorização
- Validação de dados
- Logs de requisições
- Manipulação de erros
- segurança, etc.



Benefícios de Middleware

- Evita duplicação de código de validação nos controladores
 - Facilita testes e manutenção
 - Garante integridade dos dados recebidos pela API.
-

IMPLEMENTAÇÃO



- /api
 - /control
 - ControlCargo.php
 - ControlFuncionario.php
 - /DAO
 - CargoDAO.php
 - FuncionarioDAO.php
 - /db
 - Database.php //Manipular banco de dados
 - /docs
 - //Arquivos de documentação
 - /http
 - Response.php
 - /middleware
 - CargoMiddleware.php
 - FuncionarioMiddleware.php
 - Model
 - Cargo
 - Funcionario
 - routes
 - Router.php
 - Roteador.php
 - utils
 - Logger.php
 - /sys
 - Arquivos de log e dados do sistema
- .htaccess
- app.php

Arquitetura

.htaccess

RewriteEngine on

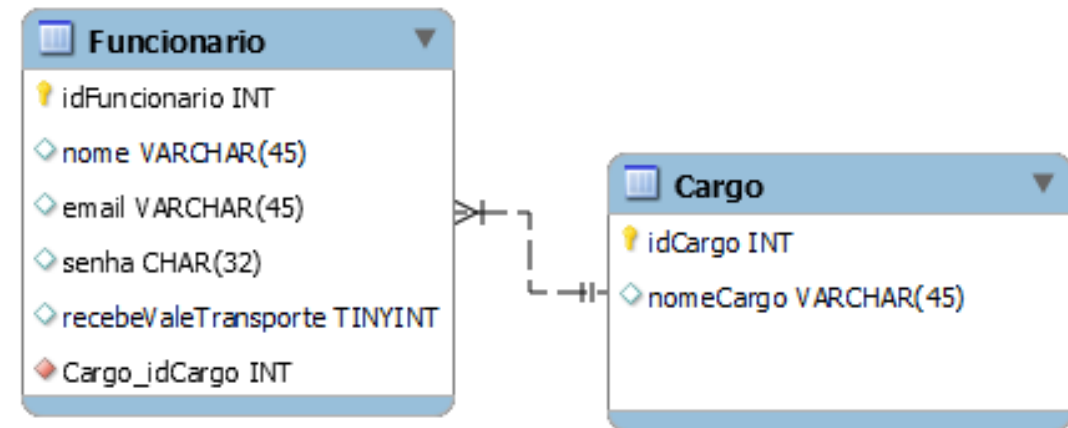
✗ Bloqueia acesso direto a arquivos .log
RewriteRule ^.*log - [F,L]

✗ Bloqueia acesso direto à pasta "api"
F = Forbidden (403), L = Last (interrompe outras regras)
RewriteRule ^api/ - [F,L]

✓ Redireciona requisições que não
#são arquivos nem diretórios para app.php
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)\$ app.php

Modelo de banco de dados

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



Classe de conexão com Sistemas Gerenciados de Banco de dados (SGBD)

- Classe: Database.php
 - Será utilizada para gerenciar a conexão com o mysql
 - Será responsável pela execução de instruções sql.

Classe: Database

Database
- static SERVER: string - static USER: string - static PASSWORD: string - static DATABASE: string - static PORT: string - static CHARACTER_SET: string - static connection: PDO
- static connect(): void + static getConnection(): void + static backup(): void

- **SERVER:** endereço do servidor mysql
- **User:** nome do usuário (root no xampp)
- **Password:** senha de acesso (“” no xampp)
- **Database:** nome do banco de dados(esquema)
- **Port:** porta de acesso(3306 no xampp)
- **CHARACTER_SET:** esquema de representação de caracteres
- **CONNECTION:** Objeto que gerencia a conexão do SGBD.
- **connect():** método que faz conexão com o servidor de banco de dados.
- **getConnection():** método que retorna uma conexão com SGBD.
- **backup():** Método que retorna todo código sql necessário para restaurar o banco de dados.

```

<?php
require_once "api/http/Response.php";
require_once "api/utils/Logger.php";
class Database {
    private const SERVER = '127.0.0.1';
    private const USER = 'root';
    private const PASSWORD = '';
    private const DATABASE = 'aula_api_2024';
    private const PORT = 3306;
    private const CHARACTER_SET = 'utf8mb4';
    // Single PDO instance (Singleton)
    private static ?PDO $connection = null;
    // Private method to establish connection
    // Public method to get connection

    public static function getConnection(): PDO
    {
        if (Database::$connection === null) {
            Database::connect();
        }
        return Database::$connection;
    }

    private static function connect(): void {
        try {
            $dsn = sprintf(
                'mysql:host=%s;port=%d;dbname=%s;charset=%s',
                Database::SERVER,
                Database::PORT,
                Database::DATABASE,
                Database::CHARACTER_SET
            );

```

```

            Database::$connection = new PDO(
                $dsn,
                Database::USER,
                Database::PASSWORD,
                [
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                    PDO::ATTR_DEFAULT_FETCH_MODE =>
PDO::FETCH_OBJ,
                    PDO::ATTR_EMULATE_PREPARES => false,
                    PDO::ATTR_PERSISTENT => false
                ]
            );
        } catch (Throwable $exception) {
            Logger::log($exception);
            (new Response(
                success: false,
                message: 'Database connection error',
                error: [
                    'errorCode' => 'database_error',
                    'message' => 'Error connecting to database',
                ],
                httpCode: 500
            ))->send();
        }
    }
}

```

ControlCargo
+ index(): never
+show(idCargo:int): never
+listPaginated(page: int, limit: int): never
* +store(stdCargo:stdClass): never *
+update(stdCargo:stdClass): never
+destroy(idCargo:int): never
+exportCSV(): never
+exportJSON(): never
+exportJSON(): never
+exportXML(): never
+importXML(\$xmlFile: array): never
+importCSV(\$csvFile: array): never
+importJson(\$jsonFile: array): never

- **index()**: lista todos os recursos (cargos)
- **show(\$idCargo)**: exhibe um recurso específico
- **store(\$cargo)**: cria um novo recurso
- **update(\$cargo)**: atualiza um recurso existente
- **destroy(\$id)**: remove um recurso
- **listPaginated(\$pagina,\$limite)**: retorna uma listagem por página
- **export...()**: Métodos que exportam dados em um determinado tipo
- **Import...()**: Método que importa dados de um determinado tipo

Middleware geralmente permite a continuação da execução do programa se o método devesse retornar verdadeiro.

Exemplo: *isvalideNomeCargo()* só permite a continuação da execução do programa se o nome é verdadeiro, caso contrário envia uma mensagem de erro para o solicitante

CargoMiddleware	
+	stringJsonToStdClass(requestBody:string): stdClass
+	isvalideNomeCargo(nomeCargo:string): self
+	hasCargoByName(nomeCargo:string): self
+	hasCargoByName(nomeCargo:string): self
+	hasNotCargoByName(nomeCargo:string): self
+	hasCargoById(idCargo:int): self
+	hasNotCargoById(idCargo:int): self
+	isValidId(id:int): self
+	isValidePage(page:int): self
+	isValideLimit(limit:int): self

- **stringJsonToStdClass(\$json):** converte o json recebido em um objeto stdClass e valida o json.
- **isvalideNomeCargo(\$nome):** continua se o nome do cago é valido
- **hasCargoByName(\$nome):** continua se existe um cargo com esse nome.
- **hasNotCargoByName(\$nome):** continua se não existe um cargo com esse nome.
- **hasCargoById(\$id):** continua se existe um cargo com esse id.
- **hasNotCargoById(\$id):** continua se não existir um cargo com esse id.
- **isValidId(\$id):** continua se o id recebido é um número válido.
- **isValidePage(\$page):** continua se a página recebido é um número de página válido.
- **isValideLimit(\$limite):** continua se o número limite é um número válido.

Rota: POST /cargos

POST >> <http://localhost:8080/cargos>

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.

The screenshot displays a REST client interface with a POST request to `http://localhost/cargos`. The request body is a JSON object: `{ "cargo": { "nomeCargo": "Auditor de dados" } }`. The response is a 200 OK status with a 12 ms latency and 121 B of data. The response body is a JSON object: `{ "success": true, "message": "Cargo cadastrado com sucesso", "data": { "cargos": { "idCargo": 12, "nomeCargo": "Auditor de dados" } } }`.

Request	Response
<pre>POST /cargos { "cargo": { "nomeCargo": "Auditor de dados" } }</pre>	<pre>200 OK 12 ms 121 B { "success": true, "message": "Cargo cadastrado com sucesso", "data": { "cargos": { "idCargo": 12, "nomeCargo": "Auditor de dados" } } }</pre>

Caminho pelas classes

App.php

1

Roteador	
+ router: Router	
2	+ __construct(router:Router)
- setup404Route(): void	
3	- setupHeaders(): void
- setupBackupRoutes(): void	
- setupImportationRoutes(): void	
- setupExportationRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupFuncionarioRoutes(): void	
4	- setupCargoRoutes(): void
- setupHeaders(): void	
- sendErrorResponse(exception:Throwable, mensagem:string): never	
5	+ start: never

CargoMiddleware	
6	+ stringJsonToStdClass(requestBody:string): stdClass
7	+ isValidNomeCargo(nomeCargo:string): self
+ hasCargoByName(nomeCargo:string): self	
+ hasCargoByName(nomeCargo:string): self	
8	+ hasNotCargoByName(nomeCargo:string): self
+ hasCargoById(idCargo:int): self	
+ hasNotCargoById(idCargo:int): self	
+ isValidId(id:int): self	
+ isValidPage(page:int): self	
+ isValidLimit(limit:int): self	

Database
- static SERVER: string - static USER: string - static PASSWORD: string - static DATABASE: string - static PORT: string - static CHARACTER_SET: string - static connection: PDO
- static connect(): void + static getConnection(): void + static backup(): void

CargDAO	
+ create(cargo: Cargo): Cargo false	11
- createWithoutId(cargo: Cargo): Cargo	12
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	
+readByPage(page: int , limit:int): array	
+readByPage(page: int , limit:int): array	
+readById(idCargo:int): Cargo	
+readByName(nomeCargo:string): Cargo null	9
+update(cargo: Cargo): Cargo null	
+delete(cargo: Cargo): bool	
+delete(cargo: Cargo): bool	

ControlCargo	
+ index(): never	
+show(idCargo:int): never	
+listPaginated(page: int, limit: int): never	
* +store(stdCargo:stdClass): never	* 10
+update(stdCargo:stdClass): never	
+destroy(idCargo:int): never	
+exportCSV(): never	
+exportJSON(): never	
+exportJSON(): never	
+exportXML(): never	
+importXML(\$xmlFile: array): never	
+importCSV(\$csvFile: array): never	
+importJson(\$jsonFile: array): never	

 Response
- success: bool - message: string - data: array - error: array - httpcode: int
+ __construct(success:bool,message:string,data:array,error:array,httpcode: int) + jsonSerialize(): array + send(): never

13

```
<?php  
require_once "api/routes/Roteador.php";  
(new Roteador())->start();
```

1

App.php

1

```
public function __construct(private $router = new Router()) {  
  
    // Cria uma nova instância do sistema de roteamento  
  
    // Essa instância será usada para registrar e tratar rotas da aplicação  
  
    $this->router = new Router();  
  
    // Configura os cabeçalhos HTTP necessários para a API funcionar corretamente  
  
    // Por exemplo: permitir requisições de diferentes domínios, definir tipo de conteúdo etc.  
    $this->setupHeaders();  
  
    // Registra todas as rotas que serão aceitas pela aplicação  
  
    // Aqui são mapeadas URLs específicas para controladores e métodos que irão tratá-las  
    $this->setupCargoRoutes();  
  
    $this->setupFuncionarioRoutes();  
  
    $this->setupExportationRoutes();  
  
    $this->setupImportationRoutes();  
  
    $this->setupBackupRoutes();  
  
    $this->setup404Route();  
  
}
```

Roteador	
	+ router: Router
2	+ __construct(router:Router)
	- setup404Route(): void
	- setupHeaders(): void
	- setupBackupRoutes(): void
	- setupImportationRoutes(): void
	- setupExportationRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupCargoRoutes(): void
	- setupHeaders(): void
	- sendErrorResponse(exception:Throwable, mensagem:string): never
	+ start: never

3

```
private function setupHeaders(): void {  
    // Permite os métodos HTTP GET, POST, PUT e DELETE nas requisições para a API  
    header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');  
  
    // Permite requisições de qualquer origem (domínio)  
  
    // O "*" significa que qualquer domínio pode acessar os recursos da API  
    header('Access-Control-Allow-Origin: *');  
  
    // (usado para enviar tokens de autenticação ou credenciais) nas requisições  
    header('Access-Control-Allow-Headers: Content-Type, Authorization');  
}
```

Roteador	
+ router: Router	
+ __construct(router:Router)	
- setup404Route(): void	
3 - setupHeaders(): void	
- setupBackupRoutes(): void	
- setupImportationRoutes(): void	
- setupExportationRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupCargoRoutes(): void	
- setupHeaders(): void	
- sendErrorResponse(exception:Throwable, mensagem:string): never	
+ start: never	

```

private function setupCargoRoutes(): void {
// Rota para criar um novo cargo
// Endpoint de exemplo: POST /cargos
// Exemplo de corpo da requisição: {"cargo": {"nomeCargo": "Analista de Sistemas"}}
$this->router->post("/cargos", function (): never {
    try {
        // Obtém o corpo da requisição (dados enviados em formato JSON)
        $requestBody = file_get_contents("php://input");
        // instância do middleware
        $middleware = new CargoMiddleware();
        // Converte a string JSON para um objeto padrão (stdClass)
        $stdObj = $middleware->stringJsonToStdClass($requestBody);
        $middleware
            ->isvalideNomeCargo($stdObj->cargo->nomeCargo)// Valida o nome do cargo
            ->hasNotCargoByName($stdObj->cargo->nomeCargo);//Verifica se não existe
            // Chama o controlador para armazenar o novo cargo no banco de dados
        (new ControlCargo())->store($stdObj);
    } catch (Throwable $exception) {
        // Caso ocorra um erro, chama a
        // função para enviar uma resposta de erro ao cliente
        $this->sendErrorResponse($exception, 'Erro ao inserir um novo cargo');
    }
    // Finaliza a execução do script após a resposta ser enviada
    exit();
});
}

```

	Roteador
	+ router: Router
	+ __construct(router:Router)
	- setup404Route(): void
	- setupHeaders(): void
	- setupBackupRoutes(): void
	- setupImportationRoutes(): void
	- setupExportationRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupFuncionarioRoutes(): void
4	- setupCargoRoutes(): void
	- setupHeaders(): void
	- sendErrorResponse(exception:Throwable, mensagem:string): never
	+ start: never

```
public function start(): void{  
    // Executa o roteador para processar a requisição e buscar a rota correspondente  
    $this->router->run();  
}
```

Roteador	
+ router: Router	
+ __construct(router:Router)	
- setup404Route(): void	
- setupHeaders(): void	
- setupBackupRoutes(): void	
- setupImportationRoutes(): void	
- setupExportationRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupCargoRoutes(): void	
- setupHeaders(): void	
- sendErrorResponse(exception:Throwable, mensagem:string): never	
5	+ start: never

```
public function stringJsonToStdClass(string $requestBody): stdClass{  
    // Decodifica o JSON para um objeto stdClass  
    $stdCargo = json_decode($requestBody);  
  
    // Verifica se houve erro na decodificação (JSON inválido)  
    if (json_last_error() !== JSON_ERROR_NONE) {  
        (new Response(  
            success: false,  
            message: 'Cargo inválido',  
            error: [  
                'codigoError' => 'validation_error',  
                'mensagem' => 'Json inválido',  
            ],  
            httpCode: 400  
        ))->send();  
        exit();  
    }  
}
```

6

CargoMiddleware

```
+ stringJsonToStdClass(requestBody:string): stdClass  
  
+ isValidNomeCargo(nomeCargo:string): self  
+ hasCargoByName(nomeCargo:string): self  
+ hasCargoByName(nomeCargo:string): self  
+ hasNotCargoByName(nomeCargo:string): self  
+ hasCargoById(idCargo:int): self  
+ hasNotCargoById(idCargo:int): self  
+ isValidId(id:int ): self  
+ isValidPage(page:int ): self  
+ isValidLimit(limit:int ): self
```

```
else if (!isset($stdCargo->cargo)) { // Verifica se o objeto "cargo" existe
    (new Response(
        success: false,
        message: 'Cargo inválido',
        error: [
            'codigoError' => 'validation_error',
            'mensagem' => 'Não foi enviado o objeto Cargo',
        ],
        httpCode: 400
    ))->send();
    exit();
}
```

CargoMiddleware	
6	+ string.JsonToStdClass(requestBody:string): stdClass
	+ isvalideNomeCargo(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
	+ hasNotCargoByName(nomeCargo:string): self
	+ hasCargoById(idCargo:int): self
	+ hasNotCargoById(idCargo:int): self
	+ isValidId(id:int): self
	+ isValidPage(page:int): self
	+ isValidLimit(limit:int): self

```
else if (!isset($stdCargo->cargo->nomeCargo)) { // Verifica se o atributo "nomeCargo" existe
    (new Response(
        success: false,
        message: 'Cargo inválido',
        error: [
            'codigoError' => 'validation_error',
            'mensagem' => 'Não foi eniado o atributo nomeCargo do Cargo',
        ],
        httpCode: 400
    ))->send();
    exit();
}

// Cria um novo objeto Cargo e preenche com os dados do JSON
$cargo = new Cargo();
$cargo->setNomeCargo($stdCargo->cargo->nomeCargo);
return $stdCargo;
}
```

6

CargoMiddleware	
	+ stringJsonToStdClass(requestBody:string): stdClass
	+ isvalideNomeCargo(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
	+ hasNotCargoByName(nomeCargo:string): self
	+ hasCargoById(idCargo:int): self
	+ hasNotCargoById(idCargo:int): self
	+ isValidId(id:int): self
	+ isValidPage(page:int): self
	+ isValidLimit(limit:int): self

```

public function isvalideNomeCargo(string $nomeCargo): self {
    // Verifica se o nome do cargo está vazio
    if (empty($nomeCargo)) {
        (new Response(
            success: false,
            message: 'Nome do cargo inválido',
            error: [
                'codigoError' => 'validation_error',
                'message' => 'O nome do cargo não pode estar vazio ou ter menos que 4 letras',
            ],
            httpCode: 400
        ))->send();
        exit(); // Interrompe a execução caso haja erro
    }
    // Verifica se o nome do cargo tem menos de 3 caracteres
    if (strlen($nomeCargo) < 3) {
        (new Response(
            success: false,
            message: 'Nome do cargo inválido',
            error: [
                'codigoError' => 'validation_error',
                'message' => 'O nome do cargo não pode estar vazio ou ter menos que 4 letras',
            ],
            httpCode: 400
        ))->send();
        exit(); // Interrompe a execução caso haja erro
    }
    // Se a validação passar, retorna o próprio objeto para permitir encadeamento
    return $this;
}

```

7

CargoMiddleware	
+ stringJsonToStdClass(requestBody:string): std	
7 + isvalideNomeCargo(nomeCargo:string): self	
+ hasCargoByName(nomeCargo:string): self	
+ hasCargoByName(nomeCargo:string): self	
+ hasNotCargoByName(nomeCargo:string): self	
+ hasCargoById(idCargo:int): self	
+ hasNotCargoById(idCargo:int): self	
+ isValidId(id:int): self	
+ isValidPage(page:int): self	
+ isValidLimit(limit:int): self	

```

public function hasNotCargoByName(string $nomeCargo): self {
    // Instancia o DAO (Data Access Object) para operações com a tabela de cargos
    $cargoDAO = new CargoDAO();
    // Busca um cargo no banco de dados pelo nome fornecido
    $cargo = $cargoDAO->readByName($nomeCargo);
    // Se nenhum cargo foi encontrado (não existe cargo com este nome)
    if (!isset($cargo)) {
        // Retorna a própria instância para permitir method chaining
        return $this;
    }
    // Se chegou aqui, significa que o cargo já existe
    // Cria e envia uma resposta de erro HTTP 400 (Bad Request)
    (new Response(
        success: false,
        message: "já existe um Cargo cadastrado com o nome ($nomeCargo)",
        error: [
            'codigoError' => 'validation_error', // Código de erro padrão
            'message' => 'Cargo cadastrado anteriormente', // Mensagem descritiva
        ],
        httpCode: 400 // Código HTTP 400 - Bad Request
    ))->send();

    // Termina a execução do script para evitar processamento adicional
    exit();
}

```

8

CargoMiddleware	
	+ string.JsonToStdClass(requestBody:string): stdClass
	+ isValidNomeCargo(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
	+ hasCargoByName(nomeCargo:string): self
8	+ hasNotCargoByName(nomeCargo:string): self
	+ hasCargoById(idCargo:int): self
	+ hasNotCargoById(idCargo:int): self
	+ isValidId(id:int): self
	+ isValidPage(page:int): self
	+ isValidLimit(limit:int): self

```

public function readByName(string $nomeCargo): Cargo|null{
    // SQL para buscar o cargo com o nome fornecido no banco de dados
    $query = 'SELECT
                idCargo,
                nomeCargo
            FROM cargo
            WHERE
                nomeCargo = :nome';
    // Prepara a query para buscar o cargo pelo nome
    $statement = Database::getConnection()->prepare(query: $query);
    // Executa a query passando o nome como parâmetro
    $statement->execute(
        params: [':nome' => $nomeCargo]
    );
    // Recupera o resultado da query
    $objStdClassLinha = $statement->fetch(mode: PDO::FETCH_OBJ);
    // Verifica se o cargo foi encontrado
    if (!$objStdClassLinha) { // Retorna null se o cargo não for encontrado
        return null;
    }
    return (new Cargo()) // Retorna um objeto Cargo com os dados encontrados
        ->setIdCargo(idCargo: $objStdClassLinha->idCargo)
        ->setNomeCargo(nomeCargo: $objStdClassLinha->nomeCargo);
}

```

9

CargoDAO	
+ create(cargo: Cargo): Cargo false	
- createWithoutId(cargo: Cargo): Cargo	
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	
+readByPage(page: int , limit:int): array	
+readByPage(page: int , limit:int): array	
+readById(idCargo:int): Cargo	
+readByName(nomeCargo:string): Cargo null	9
+update(cargo: Cargo): Cargo null	
+delete(cargo: Cargo): bool	
+delete(cargo: Cargo): bool	

10

```

public function store(stdClass $stdCargo): never {
    // Cria um novo objeto Cargo e define seu nome com base nos dados recebidos
    $cargo = new Cargo();
    $cargo->setNomeCargo($stdCargo->nomeCargo);
    // Instancia o DAO responsável por interações com o banco de dados
    $cargoDAO = new CargoDAO();
    // Chama o método 'create' do DAO para persistir o novo cargo no banco
    // e obtém o objeto já inserido (com ID gerado, por exemplo)
    $novocargo = $cargoDAO->create($cargo);
    // Monta e envia a resposta padronizada com os dados do novo cargo criado
    (new Response(
        success: true,
        message: 'Cargo cadastrado com sucesso',
        data: ['cargos' => $novocargo],
        httpCode: 200
    ))->send();
    // Finaliza a execução da aplicação imediatamente após envio da resposta
    exit();
}

```

Response
- success: bool
- message: string
- data: array
- error: array
- httpcode: int
+ __construct(success:bool,message:string,data:array,error:array,httpcode: int)
+ isonSerialize(): array
+ send(): never

13

ControlCargo
+ index(): never
+ show(idCargo:int): never
+ listPaginated(page: int, limit: int): never
+ store(stdCargo:stdClass): never
+ update(stdCargo:stdClass): never
+ destroy(idCargo:int): never
+ exportCSV(): never
+ exportJSON(): never
+ exportJSON(): never
+ exportXML(): never
+ importXML(\$xmlFile: array): never
+ importCSV(\$csvFile: array): never
+ importJson(\$jsonFile: array): never

10

```

public function create(Cargo $cargo): Cargo|false {
    // Recupera o ID do objeto Cargo
    $idCargo = $cargo->getIdCargo();

    // Caso o ID não esteja definido, realiza a inserção sem ID, permitindo que o banco gere automaticamente
    if (empty($idCargo)) {
        return $this->createWithoutId(cargo: $cargo);
    }

    // Caso o ID esteja definido, realiza a inserção manual incluindo o ID fornecido
    return $this->createWithId(cargo: $cargo);
}

```

11

POST /cargos

POST http://localhost/cargos

Params Body Auth Headers (4) Scripts Docs

JSON

```

1 {
2   "cargo": {
3     "nomeCargo": "Auditor de dados"
4   }
5 }

```

CargoDAO	
+ create(cargo: Cargo): Cargo false	11
- createWithoutId(cargo: Cargo): Cargo	
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	
+ readByPage(page: int , limit:int): array	
+ readByPage(page: int , limit:int): array	
+ readById(idCargo:int): Cargo	
+ readByName(nomeCargo:string): Cargo null	
+ update(cargo: Cargo): Cargo null	
+ delete(cargo: Cargo): bool	
+ delete(cargo: Cargo): bool	

```

private function createWithoutId(Cargo $cargo): Cargo{
    // Define a query SQL para inserir apenas o nome do cargo
    $query = 'INSERT INTO cargo (nomeCargo) VALUES (:nomeCargo)';
    // Mapeia os parâmetros da query com os valores do objeto
    $params = [
        ':nomeCargo' => $cargo->getNomeCargo()
    ];
    // Prepara e executa a instrução SQL
    $statement = Database::getConnection()->prepare(query: $query);
    $statement->execute(params: $params);
    // Obtém o ID gerado automaticamente pelo banco e atualiza o objeto
    $cargo->setIdCargo(idCargo: (int) Database::getConnection()->lastInsertId());
    // Retorna o objeto Cargo com o ID preenchido
    return $cargo;
}

```

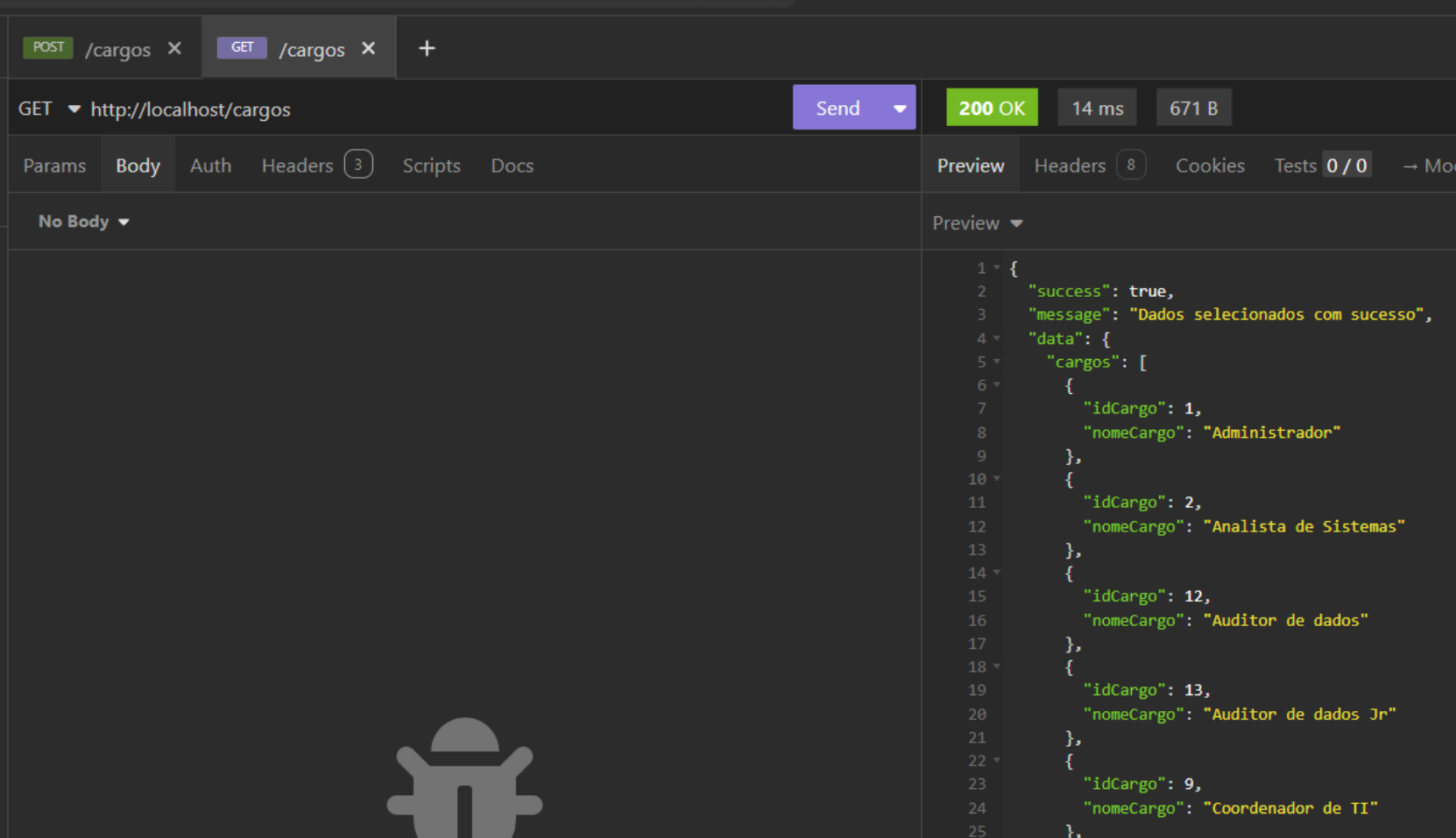
12

CargoDAO	
+ create(cargo: Cargo): Cargo false	
- createWithoutId(cargo: Cargo): Cargo	12
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	
+ readByPage(page: int , limit:int): array	
+ readByPage(page: int , limit:int): array	
+ readById(idCargo:int): Cargo	
+ readByName(nomeCargo:string): Cargo null	
+ update(cargo: Cargo): Cargo null	
+ delete(cargo: Cargo): bool	
+ delete(cargo: Cargo): bool	

Rota: GET /cargos

GET >> <http://localhost:8080/cargos>

- Ao enviar um get para um end point a api rest deve retornar a lista de recursos



The screenshot shows a REST client interface with a dark theme. At the top, there are tabs for 'POST /cargos' and 'GET /cargos'. The 'GET /cargos' tab is active. Below the tabs, the URL 'http://localhost/cargos' is entered. To the right of the URL, there is a 'Send' button. Further right, the status '200 OK' is displayed in a green box, along with '14 ms' and '671 B'. Below the URL bar, there are tabs for 'Params', 'Body', 'Auth', 'Headers' (with a count of 3), 'Scripts', and 'Docs'. The 'Body' tab is active, showing 'No Body'. To the right of the body tab, there are tabs for 'Preview', 'Headers' (with a count of 8), 'Cookies', and 'Tests' (with a count of 0/0). The 'Preview' tab is active, showing a JSON response. The JSON response is as follows:

```
1 {
2   "success": true,
3   "message": "Dados selecionados com sucesso",
4   "data": {
5     "cargos": [
6       {
7         "idCargo": 1,
8         "nomeCargo": "Administrador"
9       },
10      {
11        "idCargo": 2,
12        "nomeCargo": "Analista de Sistemas"
13      },
14      {
15        "idCargo": 12,
16        "nomeCargo": "Auditor de dados"
17      },
18      {
19        "idCargo": 13,
20        "nomeCargo": "Auditor de dados Jr"
21      },
22      {
23        "idCargo": 9,
24        "nomeCargo": "Coordenador de TI"
25      },
26    ]
27  }
28 }
```

Caminho pelas classes

App.php

1

Roteador	
+ router: Router	
2	+ __construct(router:Router)
- setup404Route(): void	
3	- setupHeaders(): void
- setupBackupRoutes(): void	
- setupImportationRoutes(): void	
- setupExportationRoutes(): void	
- setupFuncionarioRoutes(): void	
- setupFuncionarioRoutes(): void	
4	- setupCargoRoutes(): void
- setupHeaders(): void	
- sendErrorResponse(exception:Throwable, mensagem:string): never	
5	+ start: never

CargoMiddleware	
+ jsonStringToStdClass(requestBody:string): stdClass	
+ isvalideNomeCargo(nomeCargo:string): self	
+ hasCargoByName(nomeCargo:string): self	
+ hasCargoByName(nomeCargo:string): self	
+ hasNotCargoByName(nomeCargo:string): self	
+ hasCargoById(idCargo:int): self	
+ hasNotCargoById(idCargo:int): self	
+ isValidId(id:int): self	
+ isValidPage(page:int): self	
+ isValidLimit(limit:int): self	

Database
- static SERVER: string - static USER: string - static PASSWORD: string - static DATABASE: string - static PORT: string - static CHARACTER_SET: string - static connection: PDO
- static connect(): void + static getConnection(): void + static backup(): void

CargoDAO
+ create(cargo: Cargo): Cargo false - createWithoutId(cargo: Cargo): Cargo - createWithId(cargo: Cargo): Cargo false
+ readAll(): array
+readByPage(page: int , limit:int): array +readByPage(page: int , limit:int): array +readById(idCargo:int): Cargo +readByName(nomeCargo:string): Cargo null +update(cargo: Cargo): Cargo null +delete(cargo: Cargo): bool +delete(cargo: Cargo): bool

ControlCargo
+ index(): never
+show(idCargo:int): never
+listPaginated(page: int, limit: int): never
* +store(stdCargo:stdClass): never *
+update(stdCargo:stdClass): never
+destroy(idCargo:int): never
+exportCSV(): never
+exportJSON(): never
+exportJSON(): never
+exportXML(): never
+importXML(\$xmlFile: array): never
+importCSV(\$csvFile: array): never
+importJson(\$jsonFile: array): never

Response
- success: bool - message: string - data: array - error: array - httpcode: int
+ __construct(success:bool,message:string,data:array,error:array,httpcode: int) + jsonSerialize(): array
+ send(): never

Análise

- Observe que até a etapa 5 Não há mudanças na execução.
 - O item 5 há um fluxo de execução para a rota específica para tratar o GET: /cargos
- Há mudança de sequência em: 6,7 e 8

```

$this->router->get('/cargos', function (): never {
    try {
        // Obtém os parâmetros 'page' e 'limit' da URL (query string)
        // 'page' define qual página da listagem será exibida
        // 'limit' define a quantidade de registros por página
        $page = $_GET['page'] ?? null;
        $limit = $_GET['limit'] ?? null;

        // Cria uma instância do controlador de cargos para lidar com as operações
        $control = new ControlCargo();

        // Verifica se os parâmetros de paginação foram fornecidos
        if ($page !== null && $limit !== null) {
            // Se fornecidos, valida os parâmetros de página e limite utilizando o middleware
            (new CargoMiddleware())
                ->isValidPage($page) // Valida o parâmetro de página
                ->isValidLimit($limit); // Valida o parâmetro de limite

            // Se a paginação for válida, chama o método para listar os cargos paginados
            $control->listPaginated($page, $limit);
        } else {
            // Se os parâmetros de paginação não forem fornecidos, retorna todos os cargos
            $control->index();
        }
    } catch (Throwable $exception) {
        // Caso ocorra um erro, chama a função para enviar uma resposta de erro ao cliente
        $this->sendErrorResponse($exception, 'Erro na seleção de dados');
    }
    // Finaliza a execução do script após a resposta ser enviada
    exit();
});

```

	Roteador
	+ router: Router
	+ __construct(router:Router)
	- setup404Route(): void
	- setupHeaders(): void
	- setupBackupRoutes(): void
	- setupImportationRoutes(): void
	- setupExportationRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupFuncionarioRoutes(): void
4	- setupCargoRoutes(): void
	- setupHeaders(): void
	- sendErrorResponse(exception:Throwable, mensagem:string): never
	+ start: never

```

public function index(): never {

    // Cria uma instância do DAO para acessar os dados no banco

    $cargoDAO = new CargoDAO();

    // Obtém todos os cargos cadastrados

    $cargos = $cargoDAO->readAll();

    // Envia a resposta JSON com os dados encontrados

    (new Response(

        success: true,

        message: 'Dados selecionados com sucesso',

        data: ['cargos' => $cargos],

        httpCode: 200

    ))->send();

    // Encerra a execução do script

    exit();

}

```

6

ControlCargo	
+ index(): never	6
+show(idCargo:int): never	
+listPaginated(page: int, limit: int): never	
+store(stdCargo:stdClass): never	
+update(stdCargo:stdClass): never	
+destroy(idCargo:int): never	
+exportCSV(): never	
+exportJSON(): never	
+exportJSON(): never	
+exportXML(): never	
+importXML(\$xmlFile: array): never	
+importCSV(\$csvFile: array): never	
+importJson(\$jsonFile: array): never	

Response	
- success: bool	
- message: string	
- data: array	
- error: array	
- httpcode: int	
+ __construct(success:bool,message:string, data:array,error:array,httpcode: int)	
+ jsonSerialize(): array	
+ send(): never	8

```

public function readAll(): array {
    // Array que armazenará os resultados
    $resultados = [];

    // SQL para selecionar todos os cargos, ordenados por nome
    $query = 'SELECT idCargo, nomeCargo FROM cargo
        ORDER BY nomeCargo ASC';
    // Executa a query para buscar todos os cargos ordenados por nome
    $statement = Database::getConnection()->query($query);
    // Percorre cada linha retornada do banco
    // Cada linha vira um objeto stdClass
    while ($objStdClassLinha = $statement->fetch(mode: PDO::FETCH_OBJ)) {
        // Cria um novo objeto Cargo para cada registro
        $cargo = (new Cargo())
            ->setIdCargo(idCargo: $objStdClassLinha->idCargo)
            ->setNomeCargo(nomeCargo: $objStdClassLinha->nomeCargo);
        // Adiciona o objeto Cargo ao array de resultados
        $resultados[] = $cargo;
    }
    // Retorna o array de objetos Cargo
    return $resultados;
}

```

7

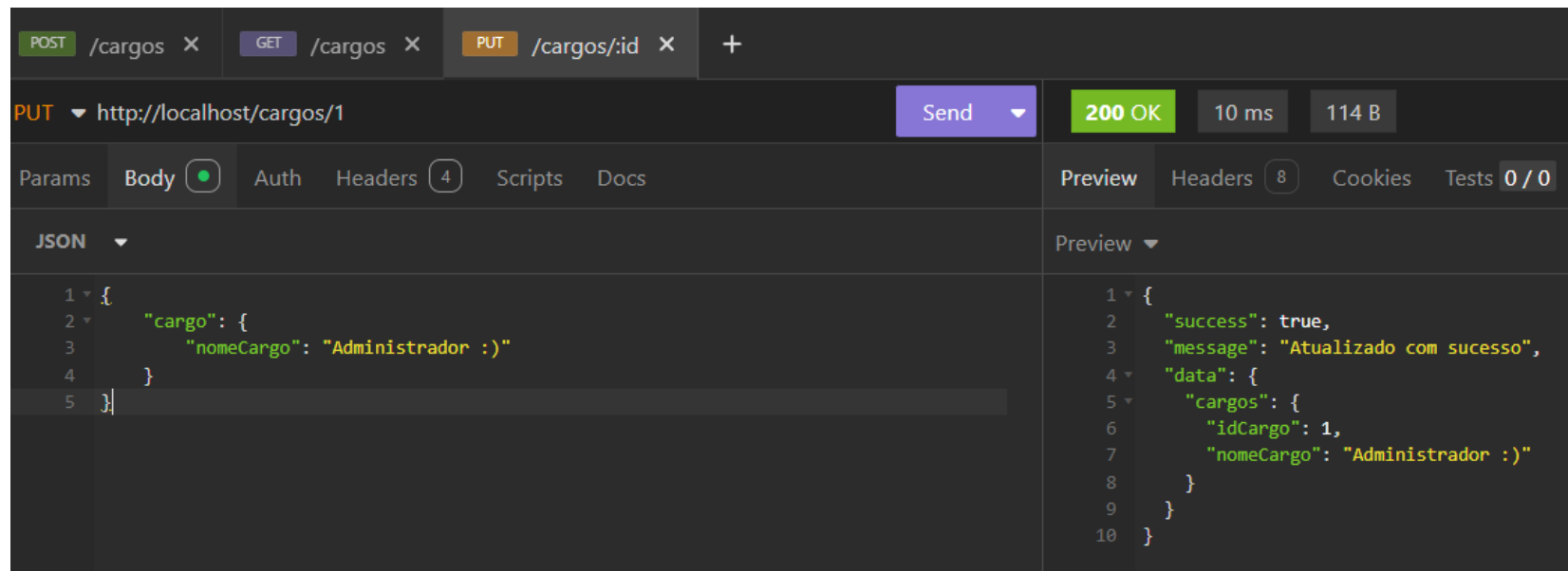
CargoDAO	
+ create(cargo: Cargo): Cargo false	
- createWithoutId(cargo: Cargo): Cargo	
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	7
+readByPage(page: int , limit:int): array	
+readByPage(page: int , limit:int): array	
+readById(idCargo:int): Cargo	
+readByName(nomeCargo:string): Cargo null	
+update(cargo: Cargo): Cargo null	
+delete(cargo: Cargo): bool	
+delete(cargo: Cargo): bool	

Rota: PUT /cargos/:idCargo

PUT >> <http://localhost:8080/cargos/1>

PUT /cargos/1

- Em uma operação de put, via uri é enviada a identificação do elemento que será alterado.
- No corpo da requisição é enviada uma string json contendo os dados que serão modificados



```

$this->router->put("/cargos/(\d+)", function (int $id): never {
    try {
        // Obtém o corpo da requisição (dados enviados em formato JSON)
        $requestBody = file_get_contents("php://input");
        // Cria uma instância do middleware para validar e processar os dados da requisição
        $middleware = new CargoMiddleware();
        // Converte a string JSON para um objeto padrão (StdClass)
        $stdObj = $middleware->stringJsonToStdClass($requestBody);
        // Valida o ID do cargo, o nome do cargo e verifica se o cargo com o nome já existe
        $middleware
            ->isValidId($id) // Valida o parâmetro ID
            ->isvalideNomeCargo($stdObj->cargo->nomeCargo) // Valida o nome do cargo
            ->hasNotCargoByName($stdObj->cargo->nomeCargo); // Verifica se o novo cargo não existe

        // Atribui o ID do cargo ao objeto para que ele seja atualizado corretamente
        $stdObj->cargo->idCargo = $id;

        // Chama o controlador para atualizar o cargo no banco de dados
        (new ControlCargo())->update($stdObj);
    } catch (Throwable $exception) {
        // Caso ocorra um erro, chama a função para enviar uma resposta de erro ao cliente
        $this->sendErrorResponse($exception, 'Erro na atualização dos dados');
    }
    // Finaliza a execução do script após a resposta ser enviada
    exit();
});

```

	Roteador
	+ router: Router
	+ __construct(router:Router)
	- setup404Route(): void
	- setupHeaders(): void
	- setupBackupRoutes(): void
	- setupImportationRoutes(): void
	- setupExportationRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupFuncionarioRoutes(): void
4	- setupCargoRoutes(): void
	- setupHeaders(): void
	- sendErrorResponse(exception:Throwable, mensagem:string): never
	+ start: never

```

public function update(stdClass $stdCargo): never{
    // Instancia o DAO responsável pelas operações de banco de dados
    $cargoDAO = new CargoDAO();
    // Cria um objeto da entidade Cargo e define os dados recebidos
    $cargo = (new Cargo())
        ->setIdCargo($stdCargo->cargo->idCargo)
        ->setNomeCargo($stdCargo->cargo->nomeCargo);
    // Tenta atualizar o cargo no banco de dados
    if ($cargoDAO->update($cargo) == true) {
        // Caso a atualização seja bem-sucedida, envia resposta com status 200
        (new Response(
            success: true,
            message: "Atualizado com sucesso",
            data: ['cargos' => $cargo],
            httpCode: 200
        ))->send();
        exit();
    } else {
        // Se a atualização falhar (ex: nome duplicado), envia resposta com status 400
        (new Response(
            success: false,
            message: "Não foi possível atualizar o cargo.",
            error: [
                'codigoError' => 'validation_error',
                'message' => 'Não é possível atualizar para um cargo que já existe',
            ],
            httpCode: 400
        ))->send();
        exit();
    }
}

```

ControlCargo	
+ index(): never	
+ show(idCargo:int): never	
+ listPaginated(page: int, limit: int): never	
+ store(stdCargo:stdClass): never	
+ update(stdCargo:stdClass): never	
+ destroy(idCargo:int): never	
+ exportCSV(): never	
+ exportJSON(): never	
+ exportJSON(): never	
+ exportXML(): never	
+ importXML(\$xmlFile: array): never	
+ importCSV(\$csvFile: array): never	
+ importJson(\$jsonFile: array): never	

```

public function update(Cargo $cargo): bool{
    // SQL para atualizar o nome do cargo no banco de dados, filtrando pelo ID do cargo
    $query = 'UPDATE cargo
              SET nomeCargo = :nomeCargo
              WHERE idCargo = :idCargo';

    // Prepara a query de atualização
    $statement = Database::getConnection()->prepare(query: $query);
    // Executa a query passando os parâmetros necessários: nome do cargo e ID do cargo
    $statement->execute(
        params: [
            ':nomeCargo' => $cargo->getNomeCargo(), // Novo nome do cargo
            ':idCargo'   => $cargo->getIdCargo()      // ID do cargo a ser atualizado
        ]
    );
    // Retorna true se pelo menos uma linha foi afetada pela atualização, indicando sucesso
    return $statement->rowCount() > 0;
}

```

CargoDAO
+ create(cargo: Cargo): Cargo false - createWithoutId(cargo: Cargo): Cargo - createWithId(cargo: Cargo): Cargo false + readAll(): array +readByPage(page: int , limit:int): array +readByPage(page: int , limit:int): array +readById(idCargo:int): Cargo +readByName(nomeCargo:string): Cargo null +update(cargo: Cargo): Cargo null +delete(cargo: Cargo): bool +delete(cargo: Cargo): bool

Rota: DELETE

/cargos/:idCargo

DELETE >> <http://localhost:8080/cargos/1>

```
// Rota para excluir um cargo específico pelo ID
// Endpoint de exemplo: DELETE /cargos/123
$this->router->delete("/cargos/(\d+)", function (int $idCargo): never {
    try {
        // Valida o ID do cargo a ser excluído
        (new CargoMiddleware())->isValidId($idCargo);

        // Chama o método do controlador para excluir o cargo com o ID específico
        (new ControlCargo())->destroy($idCargo);
    } catch (Throwable $exception) {
        // Caso ocorra um erro, chama a função para enviar uma resposta de erro ao cliente
        $this->sendErrorResponse($exception, 'Erro na exclusão de dados');
    }
    // Finaliza a execução do script após a resposta ser enviada
    exit();
});
```

	Roteador
	+ router: Router
	+ __construct(router:Router)
	- setup404Route(): void
	- setupHeaders(): void
	- setupBackupRoutes(): void
	- setupImportationRoutes(): void
	- setupExportationRoutes(): void
	- setupFuncionarioRoutes(): void
	- setupFuncionarioRoutes(): void
4	- setupCargoRoutes(): void
	- setupHeaders(): void
	- sendErrorResponse(exception:Throwable, mensagem:string): never
	+ start: never

```
public function isValidId(int $id): self{  
    // Verifica se o ID do cargo é inválido (menor ou igual a zero)  
    if ($id <= 0) {  
        (new Response(  
            success: false,  
            message: 'Identificação do cargo inválida',  
            error: [  
                'cod' => 'validation_error',  
                'message' => 'ID inválido',  
            ],  
            httpCode: 400  
        ))->send();  
  
        exit(); // Interrompe a execução caso haja erro  
    }  
    // Se a validação passar, retorna o próprio objeto para permitir encadeamento  
    return $this;  
}
```

CargoMiddleware	
+	stringJsonToStdClass(requestBody:string): stdClass
+	isvalideNomeCargo(nomeCargo:string): self
+	hasCargoByName(nomeCargo:string): self
+	hasCargoByName(nomeCargo:string): self
+	hasNotCargoByName(nomeCargo:string): self
+	hasCargoById(idCargo:int): self
+	hasNotCargoById(idCargo:int): self
+ isValidId(id:int):	self
+ isValidePage(page:int):	self
+ isValideLimit(limit:int):	self

```

public function destroy(int $idCargo): never {

    // Cria um objeto Cargo com o ID a ser excluído
    $cargo = (new Cargo())
        ->setIdCargo($idCargo);

    // Instancia o DAO responsável pelas operações no banco de dados
    $cargoDAO = new CargoDAO();

    // Verifica se a exclusão foi bem-sucedida
    if ($cargoDAO->delete($cargo) == true) {

        // 204 No Content: operação realizada com sucesso, sem conteúdo na resposta
        (new Response(httpCode: 204))->send();

    } else {

        // Envia resposta de erro caso a exclusão não tenha sido realizada
        (new Response(
            success: false,
            message: "Não foi possível excluir",
            data: ['cargos' => $cargo],
            httpCode: 400
        ))->send();

        exit();

    }

}

```

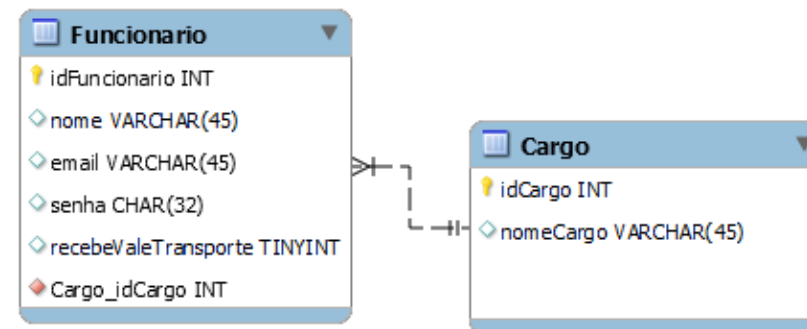


```
public function delete(Cargo $cargo): bool {  
    // SQL para deletar o cargo do banco de dados, filtrando pelo ID do cargo  
    $query = 'DELETE FROM cargo  
        WHERE idCargo = :idCargo';  
    // Prepara a query de exclusão  
    $statement = Database::getConnection()->prepare(query: $query);  
  
    // Executa a query passando o ID do cargo como parâmetro  
    $statement->execute(  
        params: [':idCargo' => $cargo->getIdCargo()]  
    );  
    // Retorna true se pelo menos uma linha foi afetada  
    return $statement->rowCount() > 0;  
}
```

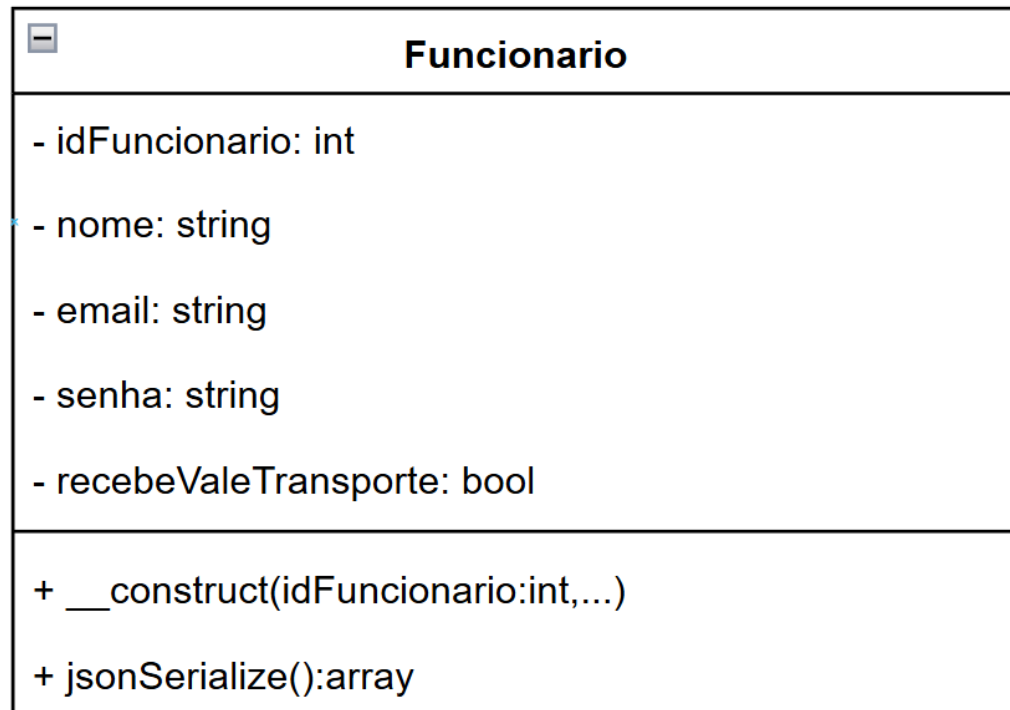
CargoDAO	
+ create(cargo: Cargo): Cargo false	
- createWithoutId(cargo: Cargo): Cargo	
- createWithId(cargo: Cargo): Cargo false	
+ readAll(): array	
+readByPage(page: int , limit:int): array	
+readByPage(page: int , limit:int): array	
+readById(idCargo:int): Cargo	
+readByName(nomeCargo:string): Cargo null	
+update(cargo: Cargo): Cargo null	
+delete(cargo: Cargo): bool	
+delete(cargo: Cargo): bool	

E quando há relacionamentos em uma tabela?

- Para implementar Funcionario na api
 - São criadas as rotas, um arquivo exclusivo de middleware, controle e dao para funcionário.
- Observe que há uma chave estrangeira em funcionário
- Algumas pequenas modificações são necessários no método ReadAll() de FuncionarioDAO e no modelo de Funcionário.



Representação simplificada



- Quando os atributos são privados, não é necessário mostrar todos os gets e sets no diagrama, é subentendido que há um get e set para cada atributo.

```
class Funcionario implements JsonSerializable{

    public function __construct(

        private ?int $idFuncionario = null,

        private string $nomeFuncionario = "",

        private string $email = "",

        private string $senha = "",

        private bool $recebeValeTransporte = false,

        private Cargo $cargo = new Cargo()

    ) {

    }

    public function jsonSerialize(): array{

        return [

            'idFuncionario' => $this->getIdFuncionario(),           // Inclui o ID do funcionário

            'nomeFuncionario' => $this->getNomeFuncionario(),       // Inclui o nome do funcionário

            'email' => $this->getEmail(),                             // Inclui o e-mail do funcionário

            'recebeValeTransporte' => $this->getRecebeValeTransporte(), // Indica se o funcionário recebe vale-transporte

            'cargo' => [

                'idCargo' => $this->cargo->getIdCargo(),             // Inclui o ID do cargo do funcionário

                'nomeCargo' => $this->cargo->getNomeCargo()          // Inclui o nome do cargo do funcionário

            ]

        ];

    }

}
```

Classe DAO de tabelas relacionadas.

- No DAO há uma modificação importante nos métodos de READ
 - É preciso fazer o Join com a tabela de relacionamento.

```
$query = 'SELECT
        idFuncionario,
        nomeFuncionario,
        email,
        recebeValeTransporte,
        idCargo,
        nomeCargo
        FROM funcionario
        JOIN cargo
        ON Cargo_idCargo = idCargo
        ORDER BY nomeFuncionario ASC';
```

```
public function readAll(): array{
    $resultados = []; // Array que armazenará os resultados
    // SQL para selecionar todos os cargos, ordenados por nome
    $query = 'SELECT
        idFuncionario,
        nomeFuncionario,
        email,
        recebeValeTransporte,
        idCargo,
        nomeCargo
    FROM funcionario
    JOIN cargo
    ON Cargo_idCargo = idCargo
    ORDER BY nomeFuncionario ASC';
    $statement = Database::getConnection()->query($query); // Executa a consula SQL
    // Percorre cada linha retornada do banco, Cada linha vira um objeto stdClass
    while ($stdLinha = $statement->fetch(mode: PDO::FETCH_OBJ)) {
        // Cria um novo objeto Cargo para cada registro
        $funcionario = (new Funcionario())
            ->setIdFuncionario(idFuncionario: $stdLinha->idFuncionario)
            ->setNomeFuncionario(nomeFuncionario: $stdLinha->nomeFuncionario)
            ->setEmail(email: $stdLinha->email)
            ->setRecebeValeTransporte(recebeValeTransporte: $stdLinha->recebeValeTransporte)
            ->getCargo()->setIdCargo(idCargo: $stdLinha->idCargo)
            ->setNomeCargo(nomeCargo: $stdLinha->nomeCargo);
        $resultados[] = $funcionario; // Adiciona o objeto Funcionario ao array de resultados
    }
    // Retorna o array de objetos Funcionario
    return $resultados;
}
```