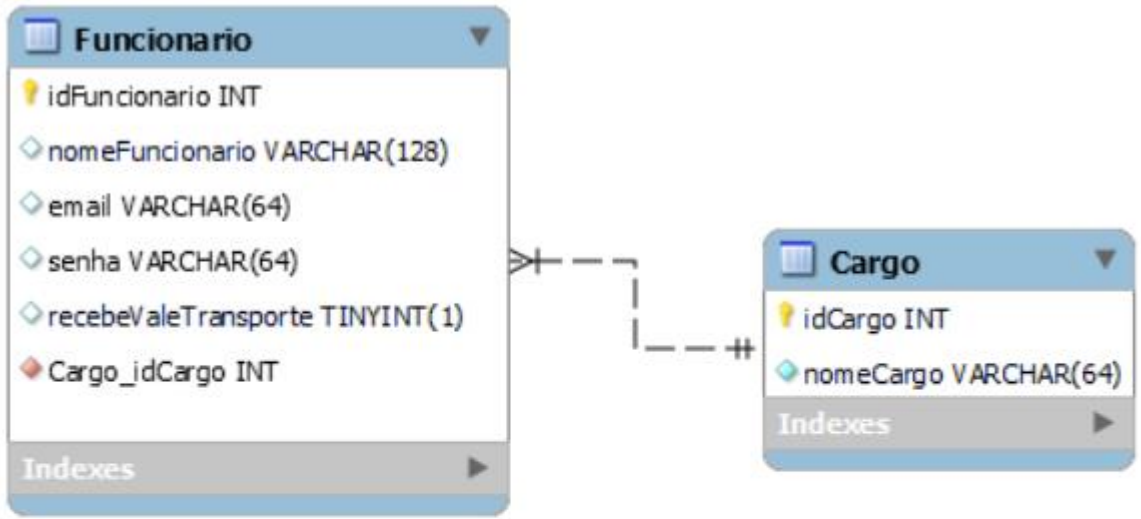
API MVC JAVASCRIPT
Cargo - Funcionário

Prof. Me. Hélio Esperidião

Pacotes necessários

- npm install jsonwebtoken --save
- npm install express --save
- npm install mysql2 --save

Modelo de banco de dados



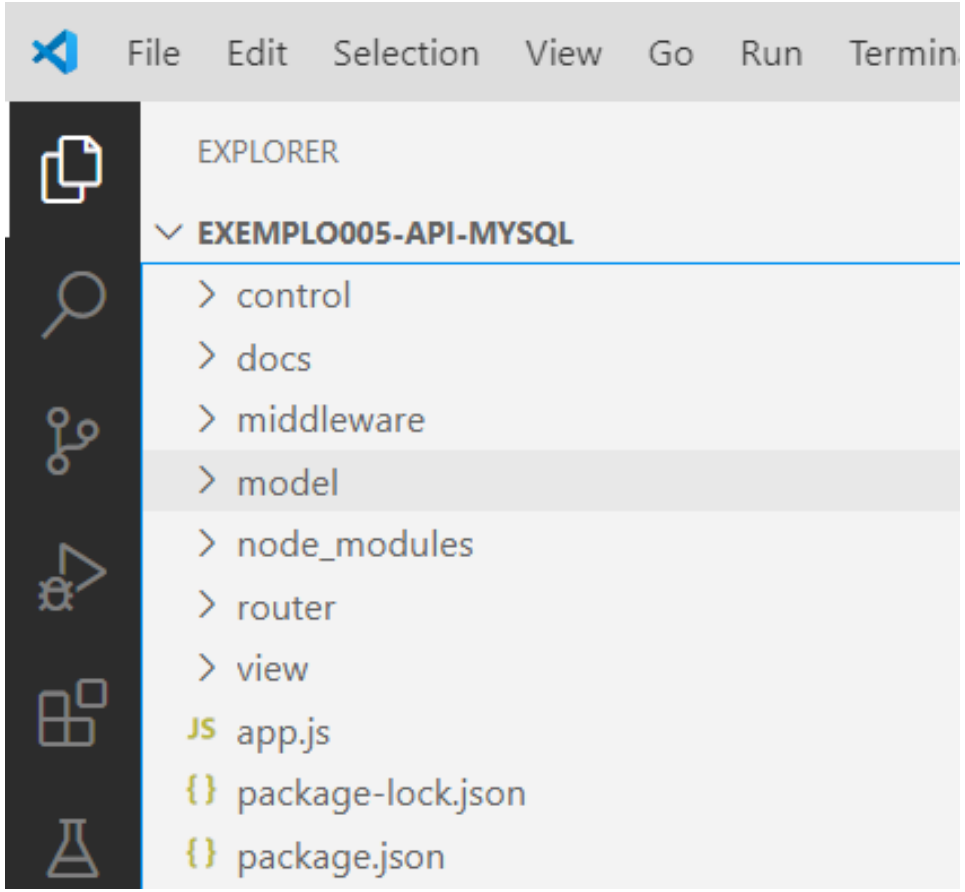
Para exportar(Database >> Foward Engineer) o modelo para os códigos sql de criação é nessário que seja mudada a versão do mysql para 5.5.6.
Vá em : Edit>> preferences >> mysql >> default target Mysql version: 5.5.6

SQL/Banco

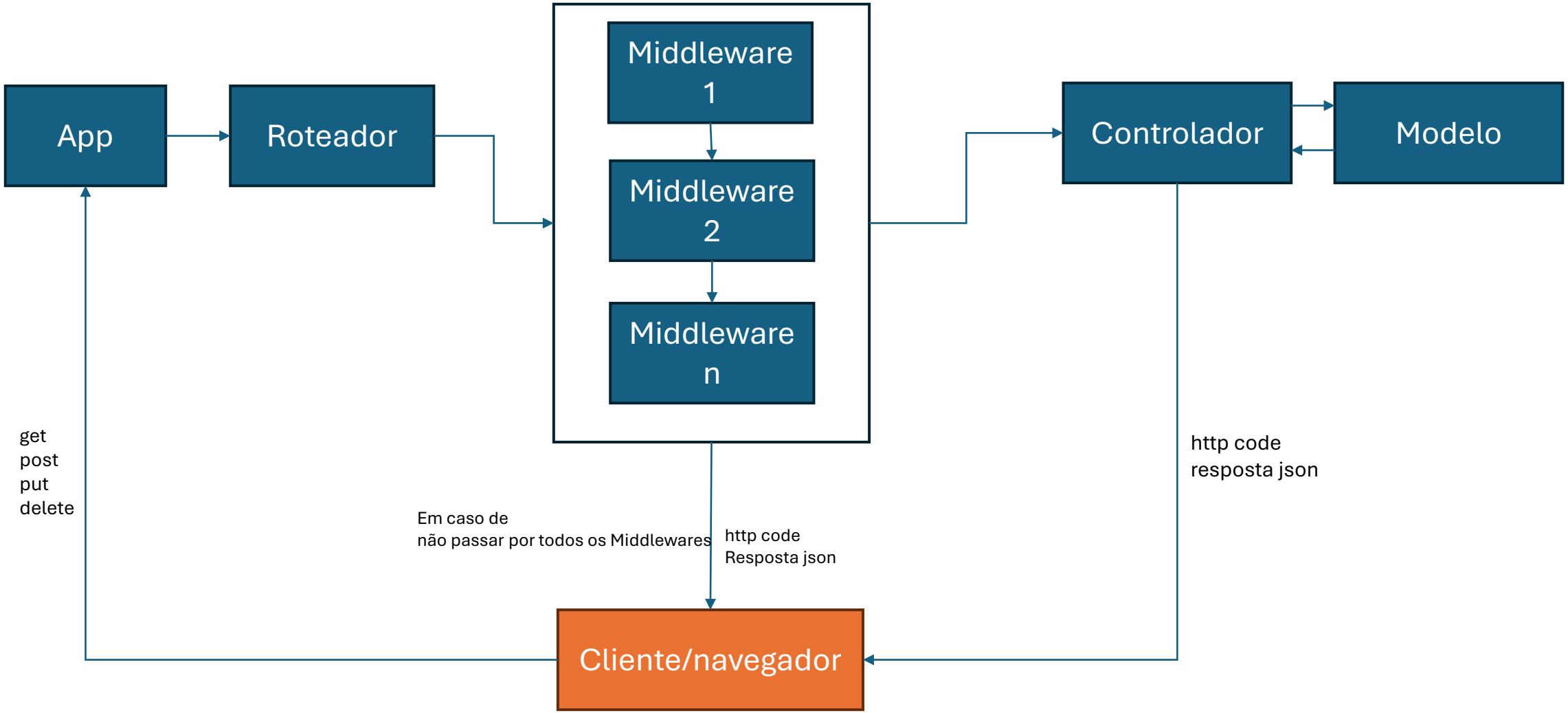
```
DROP SCHEMA  `aula_api_2024`;
CREATE SCHEMA IF NOT EXISTS `aula_api_2024` DEFAULT CHARACTER SET utf8 ;
USE `aula_api_2024` ;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Cargo` (
  `idCargo` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeCargo` VARCHAR(64) NOT NULL,
  PRIMARY KEY (`idCargo`),
  UNIQUE INDEX `idCargo_UNIQUE` (`idCargo` ASC),
  UNIQUE INDEX `nomeCargo_UNIQUE` (`nomeCargo` ASC))
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Funcionario` (
  `idFuncionario` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeFuncionario` VARCHAR(128) NULL,
  `email` VARCHAR(64) NULL,
  `senha` VARCHAR(64) NULL,
  `recebeValeTransporte` TINYINT(1) NULL,
  `Cargo_idCargo` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`idFuncionario`),
  UNIQUE INDEX `idFuncionario_UNIQUE` (`idFuncionario` ASC),
  INDEX `fk_Funcionario_Cargo_idx` (`Cargo_idCargo` ASC),
  CONSTRAINT `fk_Funcionario_Cargo`
    FOREIGN KEY (`Cargo_idCargo`)
    REFERENCES `aula_api_2024`.`Cargo` (`idCargo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (1, 'Administrador');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (2, 'Técnico em Informática Jr');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (3, 'Técnico em Informática Pleno');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (4, 'Analista de Sistemas Jr');

INSERT INTO `aula_api_2024`.`funcionario` (`nomeFuncionario`, `email`, `senha`, `recebeValeTransporte`, `Cargo_idCargo`) VALUES ('adm', 'adm@adm',
md5(123456), 1, 1);
```

Estrutura de diretórios:

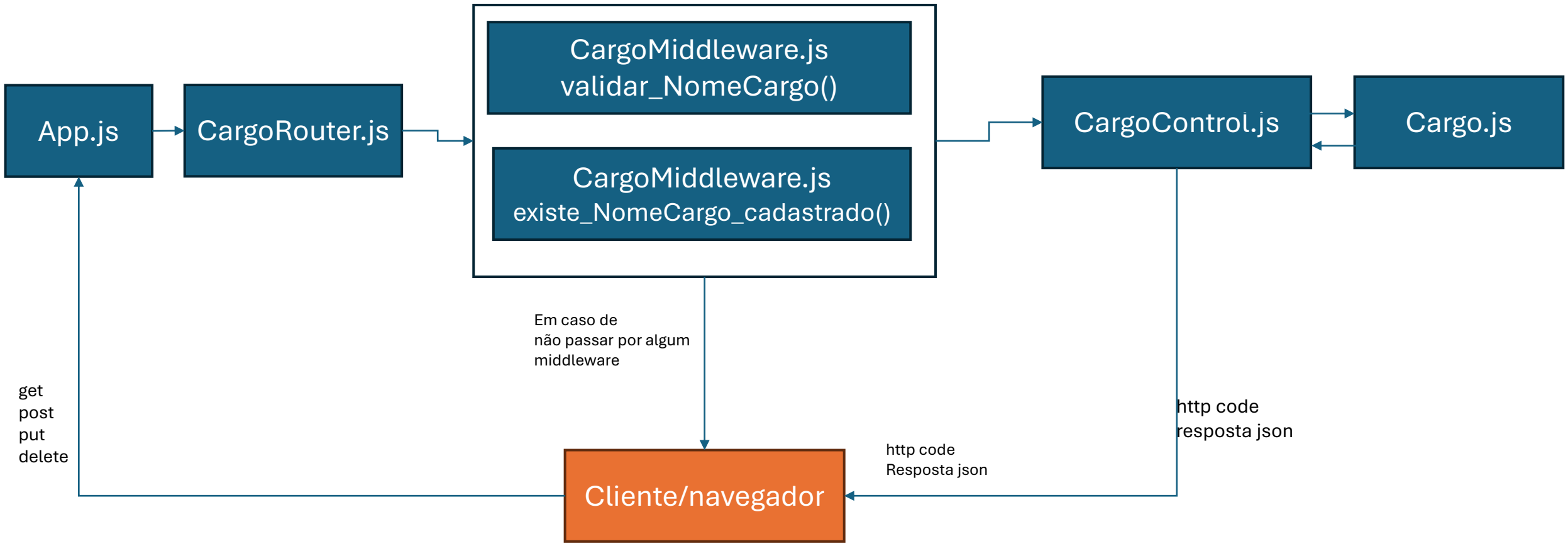
- Aplicação:
  - Control
  - Docs
  - Middleware
  - Model
  - Router
  - view



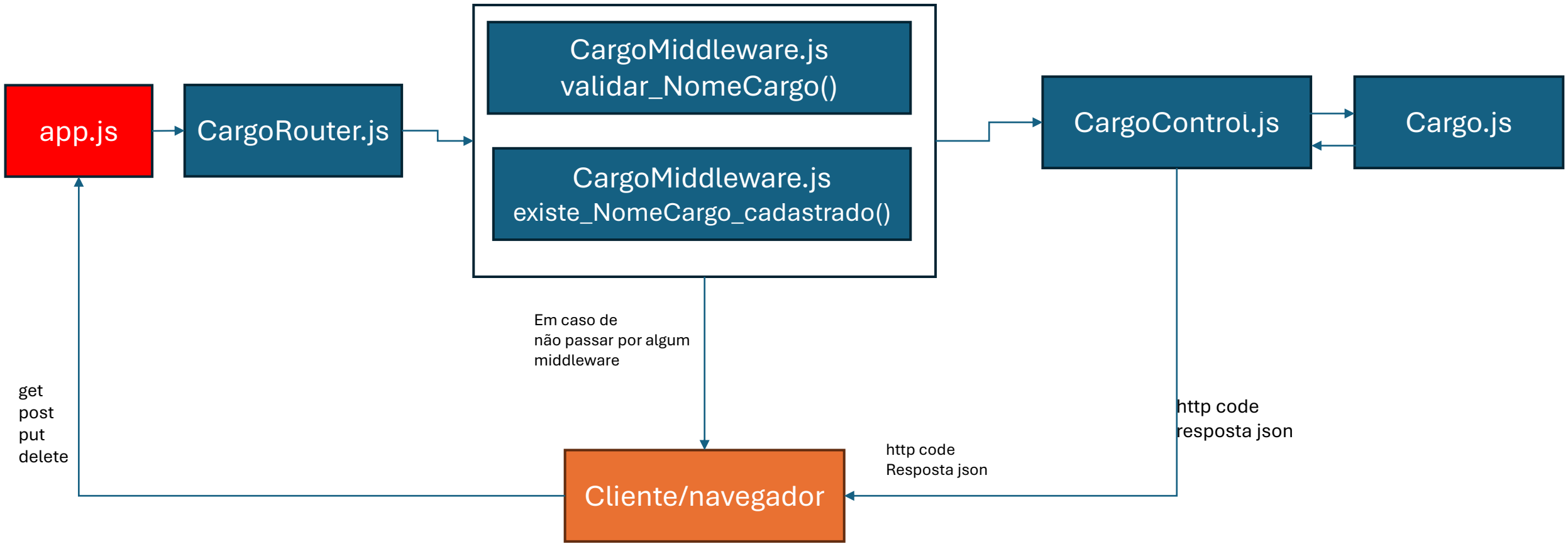
Arquitetura básica:



Arquitetura básica:



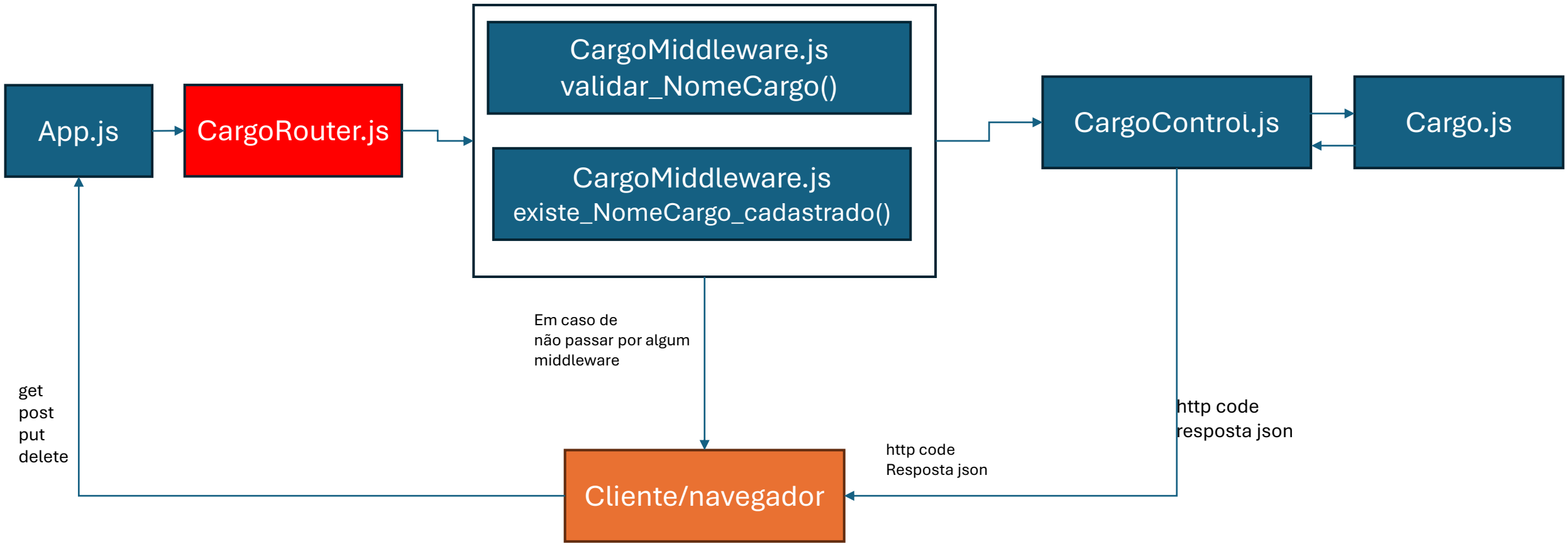
Fluxo de execução : app.js



app.js

```
// Importa o módulo 'express', que é um framework web para criar e configurar servidores HTTP.
const express = require('express');
// Importa a classe ou módulo 'CargoRouter', que provavelmente define as rotas e a lógica relacionadas a "Cargos".
const CargoRouter = require('./router/CargoRouter');
// Cria uma instância do servidor Express.
const app = express();
// Define a porta na qual o servidor vai escutar. Neste caso, a porta é 80, usada para HTTP.
const portaServico = 80;
// Adiciona um middleware que faz o parse das requisições com conteúdo JSON, permitindo que o servidor trate os dados como objetos JavaScript.
app.use(express.json());
// Cria uma instância de 'CargoRouter', responsável por definir as rotas relacionadas a cargos.
const cargoRoteador = new CargoRouter();
// Adiciona o roteador de cargos, vinculando todas as rotas que ele define ao caminho '/cargos'.
// Quando uma requisição é feita para '/cargos', as rotas definidas em 'CargoRouter' serão usadas.
app.use('/cargos',
    cargoRoteador.criarRotasCargo()
);
// Inicia o servidor, escutando na porta definida, e exibe uma mensagem no console com a URL onde o servidor está rodando.
app.listen(portaServico, () => {
    console.log(`API rodando no endereço: http://localhost:${portaServico}/`);
});
```

Fluxo de execução : CargoRouter.js



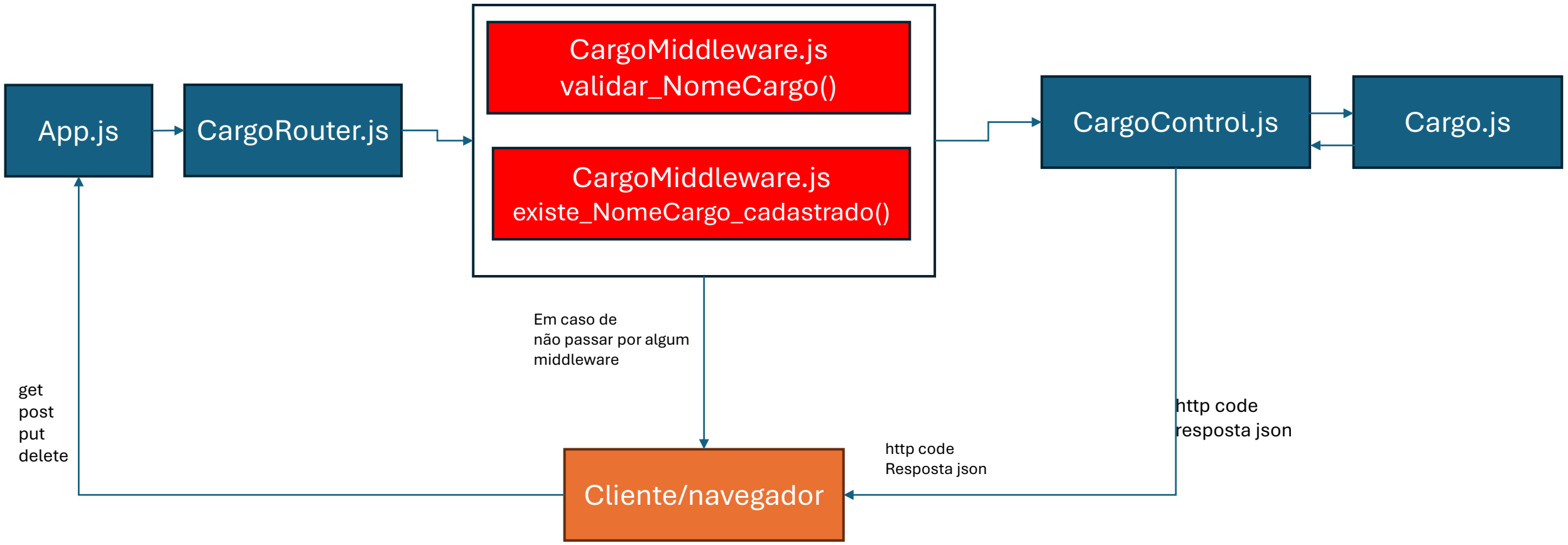
CargoRouter.js 1/2

```
// Importa o módulo 'express', que é necessário para criar o roteador.
const express = require('express');
// Importa a classe 'CargoControl', que contém a lógica de controle para as operações relacionadas a cargos.
const CargoControl = require('../control/CargoControl');
// Importa a classe 'CargoMiddleware', que contém validações e verificações antes das operações de controle.
const CargoMiddleware = require('../middleware/CargoMiddleware');
// Exporta a classe 'CargoRouter', que define as rotas relacionadas ao recurso "Cargo".
module.exports = class CargoRouter {
    // O construtor inicializa o roteador, o controle e o middleware de cargos.
    constructor() {
        // Cria uma instância do roteador do Express para definir as rotas.
        this._router = express.Router();
        // Cria uma instância de 'CargoControl', que gerencia a lógica de negócios para os cargos.
        this._cargoControl = new CargoControl();
        // Cria uma instância de 'CargoMiddleware', que valida e executa verificações antes das ações principais.
        this._cargoMiddleware = new CargoMiddleware();
    }
    // Método para criar e configurar as rotas relacionadas ao recurso "Cargo".
```

CargoRouter.js 2/2

```
criarRotasCargo() {
        // Define uma rota HTTP GET para listar todos os cargos.
        this._router.get('/',
            // Usa o método 'cargo_read_all_control' do controle para obter todos os cargos.
            this._cargoControl.cargo_read_all_control
        );
        // Define uma rota HTTP GET para buscar um cargo específico pelo ID.
        this._router.get('/:idCargo',
            // Usa o método 'cargo_read_by_id_control' do controle para obter um cargo pelo seu ID.
            this._cargoControl.cargo_read_by_id_control
        );
        // Define uma rota HTTP POST para criar um novo cargo.
        this._router.post('/',
            // Middleware para validar o nome do cargo no corpo da requisição.
            this._cargoMiddleware.validar_NomeCargo,
            // Middleware para verificar se o nome do cargo já existe no banco de dados.
            this._cargoMiddleware.existe_NomeCargo_cadastrado,
            // Usa o método 'cargo_create_control' do controle para criar um novo cargo.
            this._cargoControl.cargo_create_control
        );
        // Define uma rota HTTP DELETE para remover um cargo pelo ID.
        this._router.delete('/:idCargo',
            // Usa o método 'cargo_delete_control' do controle para deletar um cargo pelo seu ID.
            this._cargoControl.cargo_delete_control
        );
        // Define uma rota HTTP PUT para atualizar um cargo pelo ID.
        this._router.put('/:idCargo',
            // Usa o método 'cargo_update_control' do controle para atualizar os dados de um cargo específico.
            this._cargoControl.cargo_update_control
        );
        // Retorna o roteador configurado com todas as rotas para cargos.
        return this._router;
    }
}
```

Fluxo de execução : CargoMiddleware.js



CargoMiddleware.js 1/2

```
// Importa o modelo Cargo para verificar se o nome já existe no banco de dados.
const Cargo = require('../model/Cargo');
// Exporta a classe CargoMiddleware, que contém funções de validação para as requisições.
module.exports = class CargoMiddleware {
    // Método para validar o nome do cargo antes de prosseguir com a criação ou atualização.
    validar_NomeCargo(request, response, next) {
        // Recupera o nome do cargo enviado no corpo da requisição (request body).
        const nomeCargo = request.body.cargo.nomeCargo;
        // Verifica se o nome do cargo tem menos de 3 caracteres.
        if (nomeCargo.length < 3) {
            // Se o nome for inválido, cria um objeto de resposta com o status falso e a mensagem de erro.
            const objResposta = {
                status: false,
                msg: "O nome deve ter mais do que 3 letras"
            }
            // Envia a resposta com status HTTP 200 e a mensagem de erro.
            response.status(200).send(objResposta);
        } else {
            // Caso o nome seja válido, chama o próximo middleware ou a rota definida.
            next(); // Chama o próximo middleware ou rota
        }
    }
```

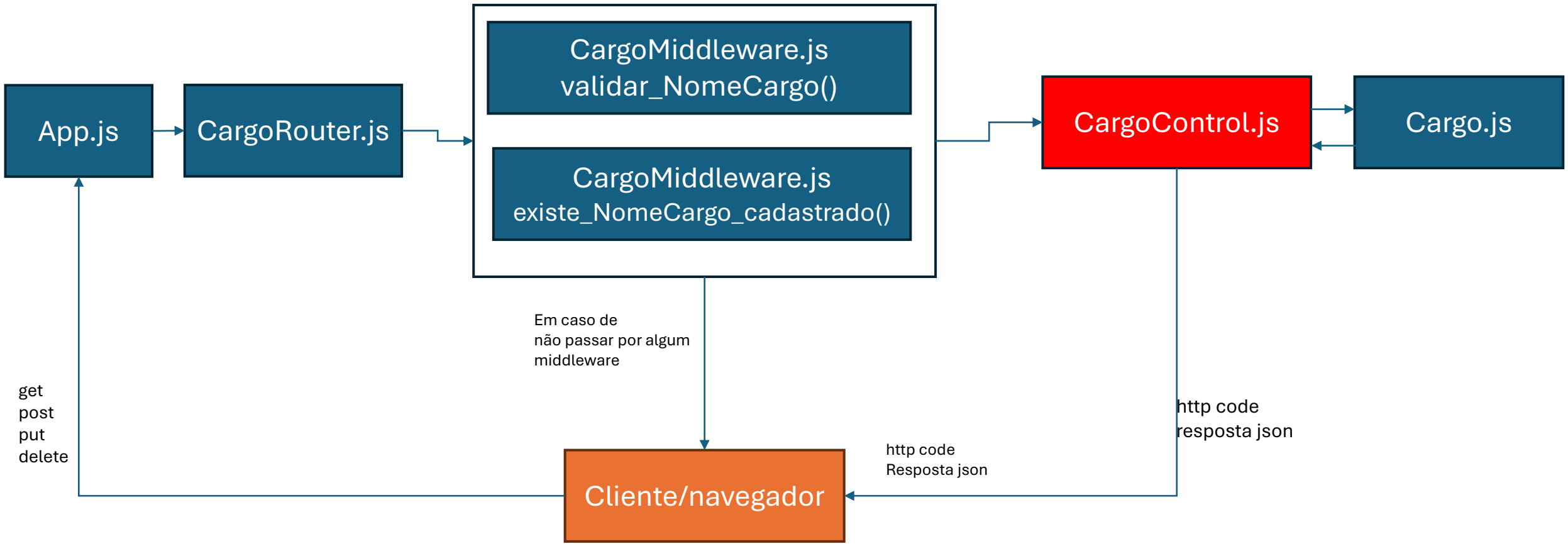
```
{
  "cargo": {
    "nomeCargo": "Novo Cargo"
  }
}
```

CargoMiddleware.js 2/2

```
// Método assíncrono para verificar se já existe um cargo com o mesmo nome cadastrado.
    async existe_NomeCargo_cadastrado(request, response, next) {
        // Recupera o nome do cargo enviado no corpo da requisição (request body).
        const nomeCargo = request.body.cargo.nomeCargo;
        // Cria uma nova instância do modelo Cargo.
        const objCargo = new Cargo();
        // Define o nome do cargo na instância do modelo.
        objCargo.nomeCargo = nomeCargo;
        // Verifica se o cargo já existe no banco de dados chamando o método isCargo().
        const cargoExiste = await objCargo.isCargo();
        // Se o cargo já existir no banco de dados, cria um objeto de resposta com o status falso e uma mensagem de erro.
        if (cargoExiste == true) {
            const objResposta = {
                status: false,
                msg: "Não é possível cadastrar um cargo com o mesmo nome de um cargo existente"
            }
            // Envia a resposta com status HTTP 200 e a mensagem de erro.
            response.status(200).send(objResposta);
        } else {
            // Caso o nome do cargo seja único, chama o próximo middleware ou rota.
            next(); // Chama o próximo middleware ou rota
        }
    }
}
```

```
{
  "cargo": {
    "nomeCargo": "Novo Cargo"
  }
}
```

Fluxo de execução : CargoControl.js



CargoControl.js
1/5

```
// Importa o módulo express para criação de APIs.
const express = require('express');
// Importa o modelo Cargo para realizar operações relacionadas à entidade Cargo.
const Cargo = require('../model/Cargo');
// Exporta a classe CargoControl, que controla as operações de CRUD (Create, Read, Update, Delete) para o Cargo.
module.exports = class CargoControl {
    // Método assíncrono para criar um novo cargo.
    async cargo_create_control(request, response) {
        // Cria uma nova instância do modelo Cargo.
        var cargo = new Cargo();
        // Atribui o nome do cargo passado no corpo da requisição (request body) à instância criada.
        cargo.nomeCargo = request.body.cargo.nomeCargo;
        // Chama o método create() do modelo Cargo para inserir o novo cargo no banco de dados.
        const isCreated = await cargo.create();
        // Cria um objeto de resposta contendo o código, status e a mensagem de sucesso ou erro.
        const objResposta = {
            cod: 1,
            status: isCreated,
            msg: isCreated ? 'Cargo criado com sucesso' : 'Erro ao criar o cargo'
        };
        // Envia a resposta HTTP com status 200 e o objeto de resposta.
        response.status(200).send(objResposta);
    }
```

```
{
  "cargo": {
    "nomeCargo": "Novo Cargo"
  }
}
```

http://localhost/cargos/106


```
// Método assíncrono para excluir um cargo existente.
    async cargo_delete_control(request, response) {
        // Cria uma nova instância do modelo Cargo.
        var cargo = new Cargo();
        // Atribui o ID do cargo passado como parâmetro na URL (request params) à instância criada.
        cargo.idCargo = request.params.idCargo;
        // Chama o método delete() do modelo Cargo para excluir o cargo do banco de dados.
        const isDeleted = await cargo.delete();
        // Cria um objeto de resposta com o código, status e a mensagem de sucesso ou erro.
        const objResposta = {
            cod: 1,
            status: isDeleted,
            msg: isDeleted ? 'Cargo excluído com sucesso' : 'Erro ao excluir o cargo'
        };
        // Envia a resposta HTTP com status 200 e o objeto de resposta.
        response.status(200).send(objResposta);
    }
```

http://localhost/cargos/106

CargoControl.js
3/5

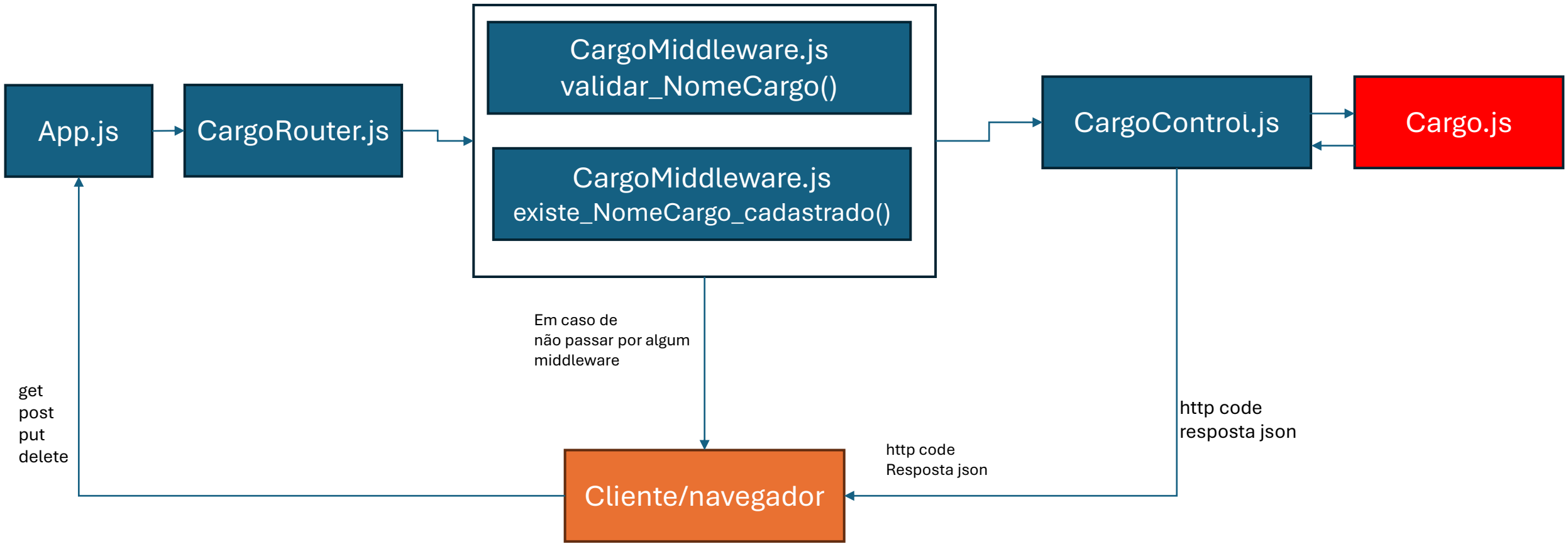
```
// Método assíncrono para atualizar um cargo existente.
async cargo_update_control(request, response) {
    // Cria uma nova instância do modelo Cargo.
    var cargo = new Cargo();
    // Atribui o ID e o nome do cargo passados na URL e no corpo da requisição, respectivamente.
    cargo.idCargo = request.params.idCargo;
    cargo.nomeCargo = request.body.cargo.nomeCargo;
    // Chama o método update() do modelo Cargo para atualizar o cargo no banco de dados.
    const isUpdated = await cargo.update();
    // Cria um objeto de resposta com o código, status e a mensagem de sucesso ou erro.
    const objResposta = {
        cod: 1,
        status: true,
        msg: isUpdated ? 'Cargo atualizado com sucesso' : 'Erro ao atualizar o cargo'
    };
    // Envia a resposta HTTP com status 200 e o objeto de resposta.
    response.status(200).send(objResposta);
}
```

```
{
  "cargo":{
    "nomeCargo": "Novo Cargo"
  }
}
```

```
// Método assíncrono para obter todos os cargos.
async cargo_read_all_control(request, response) {
    // Cria uma nova instância do modelo Cargo.
    var cargo = new Cargo();
    // Chama o método readAll() para buscar todos os cargos no banco de dados.
    const resultado = await cargo.readAll();
    // Cria um objeto de resposta contendo o código, status, mensagem e a lista de cargos.
    const objResposta = {
        cod: 1,
        status: true,
        msg: 'Executado com sucesso',
        cargos: resultado
    };
    // Envia a resposta HTTP com status 200 e o objeto de resposta.
    response.status(200).send(objResposta);
}
```

```
    // Método assíncrono para obter um cargo pelo ID.
    async cargo_read_by_id_control(request, response) {
        // Cria uma nova instância do modelo Cargo.
        var cargo = new Cargo();
        // Atribui o ID do cargo passado como parâmetro na URL (request params) à instância criada.
        cargo.idCargo = request.params.idCargo;

        // Chama o método readByID() para buscar o cargo pelo ID no banco de dados.
        const resultado = await cargo.readByID();
        // Cria um objeto de resposta contendo o código, status, mensagem e o cargo encontrado (ou não).
        const objResposta = {
            cod: 1,
            status: true,
            msg: resultado ? 'Cargo encontrado' : 'Cargo não encontrado',
            cargo: resultado
        };
        // Envia a resposta HTTP com status 200 e o objeto de resposta.
        response.status(200).send(objResposta);
    }
};
```

Fluxo de execução : Cargo.js



Cargo.js

```
// Importa o módulo Banco para realizar conexões com o banco de dados.
const Banco = require('./Banco');

// Define a classe Cargo para representar a entidade Cargo.
class Cargo {
    // Construtor da classe Cargo que inicializa as propriedades.
    constructor() {
        this._idCargo = null;  // ID do cargo, inicialmente nulo.
        this._nomeCargo = '';  // Nome do cargo, inicialmente uma string vazia.
    }
```

Cargo.js

```
// Método assíncrono para criar um novo cargo no banco de dados.
async create() {
    const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
    const SQL = 'INSERT INTO cargo (nomeCargo) VALUES (?);';  // Query SQL para inserir o nome do cargo.

    try {
        const [result] = await conexao.promise().execute(SQL, [this._nomeCargo]);  // Executa a query.
        this._idCargo = result.insertId;  // Armazena o ID gerado pelo banco de dados.
        return result.affectedRows > 0;  // Retorna true se a inserção afetou alguma linha.
    } catch (error) {
        console.error('Erro ao criar o cargo:', error);  // Exibe erro no console se houver falha.
        return false;  // Retorna false caso ocorra um erro.
    }
}
```

Cargo.js

```
// Método assíncrono para excluir um cargo do banco de dados.
async delete() {
    const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
    const SQL = 'DELETE FROM cargo WHERE idCargo = ?;';  // Query SQL para deletar um cargo pelo ID.

    try {
        const [result] = await conexao.promise().execute(SQL, [this._idCargo]);  // Executa a query de exclusão.
        return result.affectedRows > 0;  // Retorna true se alguma linha foi afetada (cargo deletado).
    } catch (error) {
        console.error('Erro ao excluir o cargo:', error);  // Exibe erro no console se houver falha.
        return false;  // Retorna false caso ocorra um erro.
    }
}
```

Cargo.js

```
// Método assíncrono para atualizar os dados de um cargo no banco de dados.
async update() {
    const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
    const SQL = 'UPDATE cargo SET nomeCargo = ? WHERE idCargo = ?;';  // Query SQL para atualizar o nome de um cargo.

    try {
        const [result] = await conexao.promise().execute(SQL, [this._nomeCargo, this._idCargo]);  // Executa a query de atualização.
        return result.affectedRows > 0;  // Retorna true se a atualização afetou alguma linha.
    } catch (error) {
        console.error('Erro ao atualizar o cargo:', error);  // Exibe erro no console se houver falha.
        return false;  // Retorna false caso ocorra um erro.
    }
}
```

Cargo.js

```
// Método assíncrono para verificar se um cargo já existe no banco de dados.
async isCargo() {
    const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
    const SQL = 'SELECT COUNT(*) AS qtd FROM cargo WHERE nomeCargo = ?;';  // Query SQL para contar cargos com o mesmo nome.

    try {
        const [rows] = await conexao.promise().execute(SQL, [this._nomeCargo]);  // Executa a query.
        return rows[0].qtd > 0;  // Retorna true se houver algum cargo com o mesmo nome.
    } catch (error) {
        console.error('Erro ao verificar o cargo:', error);  // Exibe erro no console se houver falha.
        return false;  // Retorna false caso ocorra um erro.
    }
}
```

Cargo.js

```
// Método assíncrono para ler todos os cargos do banco de dados.
async readAll() {
    const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
    const SQL = 'SELECT * FROM cargo ORDER BY nomeCargo;';  // Query SQL para selecionar todos os cargos ordenados pelo nome.

    try {
        const [rows] = await conexao.promise().execute(SQL);  // Executa a query de seleção.
        return rows;  // Retorna a lista de cargos.
    } catch (error) {
        console.error('Erro ao ler cargos:', error);  // Exibe erro no console se houver falha.
        return [];  // Retorna uma lista vazia caso ocorra um erro.
    }
}
```

Cargo.js

```
// Método assíncrono para ler um cargo pelo seu ID.
 async readByID() {
     const conexao = Banco.getConexao();  // Obtém a conexão com o banco de dados.
     const SQL = 'SELECT * FROM cargo WHERE idCargo = ?;';  // Query SQL para selecionar um cargo pelo ID.

     try {
         const [rows] = await conexao.promise().execute(SQL, [this._idCargo]);  // Executa a query de seleção.
         return rows;  // Retorna o cargo correspondente ao ID.
     } catch (error) {
         console.error('Erro ao ler cargo pelo ID:', error);  // Exibe erro no console se houver falha.
         return null;  // Retorna null caso ocorra um erro.
     }
 }
```

Cargo.js

```
  // Getter para obter o valor de idCargo.
    get idCargo() {
        return this._idCargo;
    }

    // Setter para definir o valor de idCargo.
    set idCargo(idCargo) {
        this._idCargo = idCargo;
    }

    // Getter para obter o valor de nomeCargo.
    get nomeCargo() {
        return this._nomeCargo;
    }

    // Setter para definir o valor de nomeCargo.
    set nomeCargo(nomeCargo) {
        this._nomeCargo = nomeCargo;
    }
}

// Exporta a classe Cargo para que possa ser utilizada em outros módulos.
module.exports = Cargo;
```

Token JWT

- npm install jsonwebtoken --save

MeuTokenJWT.js 1/5

```
const jwt = require('jsonwebtoken');

class MeuTokenJWT {
    constructor() {
        this._key = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3"; // Chave secreta
        this._alg = 'HS256'; // Algoritmo de criptografia
        this._type = 'JWT';
        this._iss = 'http://localhost'; // Emissor do token
        this._aud = 'http://localhost'; // Destinatário do token
        this._sub = "acesso_sistema"; // Assunto do token
        this._duracaoToken = 3600 * 24 * 30; // Duração do token (30 dias)
    }
```

MeuTokenJWT.js 2/5

```
gerarToken(parametroClaims) {
        const headers = {
            alg: this._alg,
            typ: this._type
        };
        const payload = {
            iss: this._iss, // Emissor do token
            aud: this._aud, // Destinatário do token
            sub: this._sub, // Assunto do token
            iat: Math.floor(Date.now() / 1000), // Momento de criação (em segundos)
            exp: Math.floor(Date.now() / 1000) + this._duracaoToken, // Expiração (em segundos)
            nbf: Math.floor(Date.now() / 1000), // Não é válido antes do tempo especificado
            jti: require('crypto').randomBytes(16).toString('hex'), // Identificador único (jti)
            email: parametroClaims.email, // Claims públicas
            role: parametroClaims.role,
            name: parametroClaims.name,
            idFuncionario: parametroClaims.idFuncionario // Claims privadas
        };
        // Gera o token utilizando a biblioteca jsonwebtoken
        const token = jwt.sign(payload, this._key, { algorithm: this._alg, header: headers });
        return token;
    }
```

MeuTokenJWT.js 3/5

```
gerarToken(parametroClaims) {
        const headers = {
            alg: this._alg,
            typ: this._type
        };
        const payload = {
            iss: this._iss, // Emissor do token
            aud: this._aud, // Destinatário do token
            sub: this._sub, // Assunto do token
            iat: Math.floor(Date.now() / 1000), // Momento de criação (em segundos)
            exp: Math.floor(Date.now() / 1000) + this._duracaoToken, // Expiração (em segundos)
            nbf: Math.floor(Date.now() / 1000), // Não é válido antes do tempo especificado
            jti: require('crypto').randomBytes(16).toString('hex'), // Identificador único (jti)
            email: parametroClaims.email, // Claims públicas
            role: parametroClaims.role,
            name: parametroClaims.name,
            idFuncionario: parametroClaims.idFuncionario // Claims privadas
        };
        // Gera o token utilizando a biblioteca jsonwebtoken
        const token = jwt.sign(payload, this._key, { algorithm: this._alg, header: headers });
        return token;
    }
```

MeuTokenJWT.js 4/5

```
validarToken(stringToken) {
    if (!stringToken || stringToken.trim() === "") {
        return false;
    }
    const token = stringToken.replace("Bearer ", "").trim();
    try {
        const decoded = jwt.verify(token, this._key, { algorithms: [this._alg] });
        this.payload = decoded;
        return true;
    } catch (err) {
        // Lidar com diferentes tipos de erro
        if (err instanceof jwt.TokenExpiredError) {
            console.error("Token expirado");
        } else if (err instanceof jwt.JsonWebTokenError) {
            console.error("Token inválido");
        } else {
            console.error("Erro geral", err);
        }
        return false;
    }
}
```

MeuTokenJWT.js 5/5

```
    getPayload() {
        return this.payload;
    }
    setPayload(payload) {
        this.payload = payload;
    }
    getAlg() {
        return this._alg;
    }
    setAlg(alg) {
        this._alg = alg;
    }
}
module.exports = MeuTokenJWT;
```

Onde usar a classe MeuTokenJWT?

- Onde posicionar?
- Onde utilizar?