MIDDLEWARES NO EXPRESS.JS

O QUE É UM MIDDLEWARE?

No contexto do Express, um **middleware** é qualquer função que tenha a assinatura:

```
1 (req, res, next) ⇒ { ... }
```

Essa função é **executada entre a requisição recebida e a resposta enviada**. Ela pode:

- Modificar **req** ou **res**
- Encerrar a resposta com **res.send()** / **res.json()** etc.
- Passar o controle adiante com **next()**
- Lidar com erros se tiver a assinatura **(err, req, res, next)**

O CICLO DE VIDA DE UMA REQUISIÇÃO

Cada requisição percorre um **pipeline** de middlewares. Visualize como um fluxo:

```
-> Requisição entra
  -> Middleware 1
    -> Middleware 2
      -> Middleware 3
        -> Handler da Rota
          -> Middleware de erro (se houver falha)
-> Resposta enviada
```

TIPOS DE MIDDLEWARES NO EXPRESS

1 - **Globais (Application-Level)**

Executados em toda requisição:

```
app.use(express.json()); // built-in middleware
app.use((req, res, next) => {
  console.log(`[${req.method}] ${req.url}`);
  next();
});
```

2 - **Por Rota**

Definidos diretamente em uma rota específica:

```
app.get('/admin', autenticarAdmin, (req, res) => {
  res.send('Área de administração');
});
```

TIPOS DE MIDDLEWARES NO EXPRESS

3 - **Middlewares de Erro**

Identificados pela presença do primeiro argumento **err**:

```
1 app.use((err, req, res, next) ⇒ {
2   console.error(err.stack);
3   res.status(500).send('Erro interno no servidor');
4 });
```

*Importante: só é chamado se algum middleware anterior passar o erro para **next(err)**.*

4 - **Middlewares de Terceiros**

Ex: **morgan**, **cors**, **helmet**, **express-rate-limit**

```
1 const morgan = require('morgan');
2 app.use(morgan('dev'));
```

CASOS DE USO COMUNS

Caso de uso	Middleware
Log de requisições	`morgan` ou middleware customizado
Autenticação	Token JWT verificado em middleware
Validação de dados	`express-validator`, Zod, Yup etc.
Erros globais	`app.use(tratadorDeErro)`
Regras de negócios por role	Middleware condicional baseado em `req.user.role`
Rate limiting	`express-rate-limit`
Segurança	`helmet`, `cors`, `xss-clean`

EXEMPLO AVANÇADO: PIPELINE COM VALIDAÇÃO E AUTENTICAÇÃO

```
const express = require('express');
const app = express();

function validarEmail(req, res, next) {
  const { email } = req.body;
  if (!email.includes('@')) {
    return res.status(400).json({ erro: 'Email inválido' });
  }
  next();
}

function autenticarToken(req, res, next) {
  const token = req.headers.authorization?.split(' ')[1];
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.usuario = payload;
    next();
  } catch {
    res.status(401).json({ erro: 'Token inválido' });
  }
}

app.post('/perfil', validarEmail, autenticarToken, (req, res) => {
  res.json({ msg: `Olá, ${req.usuario.nome}` });
});

```

ARQUITETURA ESCALÁVEL COM MIDDLEWARES

Organize seus middlewares por responsabilidade:

```
middlewares/
│
├── auth/
│   └── verificarToken.js
│
├── validation/
│   └── validarUsuario.js
│
├── error/
│   └── tratadorGlobal.js
```

ARQUITETURA ESCALÁVEL COM MIDDLEWARES

E importe nas rotas:

```
1 const verificarToken = require('./middlewares/auth/verificarToken');
2 const validarUsuario = require('./middlewares/validation/validarUsuario');
3
4 app.post('/usuarios', validarUsuario, verificarToken, controlador.criarUsuario);
```

Padrões e Boas Práticas

- ✓ Crie middlewares puros, sem acoplamento com regras de negócio
- ✓ Nunca esqueça de chamar next() se não for encerrar a resposta
- ✓ Use middlewares para garantir segurança e validação
- ✓ Mantenha middlewares com responsabilidade única
- ✓ Crie middlewares reutilizáveis por contexto (auth, rate, validate etc.)

ARMADILHAS COMUNS

Erro	Causa
Não chamar `next()`	A requisição nunca chega na rota final
Encerrar a resposta sem `return`	Código abaixo de `res.send()` ainda pode ser executado
Misturar lógica de negócio	Middleware vira uma rota escondida

TESTANDO MIDDLEWARES

Use ferramentas como **supertest** e crie mocks de **req**, **res**, e **next()**:

```
1  const { validarEmail } = require('./middlewares/validarEmail');
2
3  test('Deve retornar erro se email for inválido', () ⇒ {
4    const req = { body: { email: 'invalido' } };
5    const res = { status: jest.fn().mockReturnThis(), json: jest.fn() };
6    const next = jest.fn();
7
8    validarEmail(req, res, next);
9
10   expect(res.status).toHaveBeenCalledWith(400);
11 });
12
```

THANK YOU

@wallace027dev