

Explicação Detalhada

📌 1. Importações e dependências

typescript



Copy



Edit

```
import { Request, Response, NextFunction } from 'express'; import jwt from 'jsonwebtoken'; import { z } from 'zod';
```

- `express` → Framework para criar APIs REST.
 - `Request` : Representa a requisição feita pelo cliente.
 - `Response` : Representa a resposta enviada ao cliente.
 - `NextFunction` : Permite passar para o próximo middleware da pilha.
- `jsonwebtoken` → Biblioteca para manipular **JWTs (JSON Web Tokens)**.
 - Usamos `jwt.verify()` para validar tokens recebidos.
- `zod` → Biblioteca de **validação de dados**.
 - Usamos para garantir que o cabeçalho HTTP contenha um token válido.

📌 2. Definição do esquema de validação (`authSchema`)

typescript



Copy



Edit

```
const authSchema = z.object({ headers: z.object({ authorization: z.string().refine((val) => val.startsWith('Bearer '), { message: 'Token deve começar com Bearer', }), }), });
```

🔧 O que esse código faz?

Ele **valida** os cabeçalhos (`headers`) da requisição para garantir que:

1. Existe um cabeçalho chamado `authorization` .
2. O valor do cabeçalho é uma `string` .
3. A string começa com **"Bearer "** (padrão para tokens JWT).

Se alguma dessas condições falhar, o `Zod` **lançará um erro automaticamente**.

❌ Exemplos de valores inválidos:

Cabeçalho Authorization

Erro retornado

null

"Token inválido"

"12345"

"Token deve começar com Bearer"

"Bearer"

"Token inválido"

📌 3. Middleware de Autenticação (authMiddleware)

typescript



Copy



Edit

```
export const authMiddleware = async (req: Request, res: Response, next: NextFunction) => {
```

- Esse middleware será usado **antes de rotas protegidas** para verificar se o usuário está autenticado.
- Ele recebe:
 - req : A requisição HTTP.
 - res : A resposta HTTP.
 - next : Função para chamar o próximo middleware (ou controlador da rota).

📌 4. Validação do Token

typescript



Copy



Edit

```
try { const { headers } = authSchema.parse(req); const token = headers.authorization.split(' ')[1];
```

🔧 O que acontece aqui?

1. authSchema.parse(req) :

- Valida os **headers** da requisição com `zod`.
- Se falhar, gera um erro.

2. `headers.authorization.split(' ')[1]` :

- O cabeçalho vem no formato:

```
makefile
```



Copy



Edit

```
Authorization: Bearer eyJhbGciOiJIUzI1...
```

- O código divide a string em **duas partes**:
 - `"Bearer"` (posição `[0]`)
 - **Token JWT real** (posição `[1]`)
- O `split(' ')[1]` pega apenas o token.

3. Se token **estiver vazio**, retorna erro:

```
typescript
```



Copy



Edit

```
if (!token) { return res.status(401).json({ message: 'Token não fornecido' }); }
```

- Código `401 Unauthorized` → Significa que a autenticação falhou.

5. Decodificação e Verificação do Token

```
typescript
```



Copy



Edit

```
const decoded = jwt.verify(token, process.env.JWT_SECRET || 'your-secret-key'); req.user = decoded; next();
```

Explicação

- `jwt.verify(token, chave_secreta)` :
 - Verifica se o token é válido.
 - Se o token for inválido ou expirado, lança um erro.
- `process.env.JWT_SECRET || 'your-secret-key'` :
 - A chave usada para assinar/verificar o token.

- **Melhor prática** → Usar uma variável de ambiente (`process.env.JWT_SECRET`) ao invés de uma chave fixa.
- `req.user = decoded;`
 - **Armazena os dados do usuário** decodificados no `req.user` .
 - Isso permite acessar o usuário autenticado em outras partes da aplicação.
- `next();`
 - **Chama o próximo middleware** ou controlador da rota.

📌 6. Tratamento de Erros

typescript



Copy



Edit

```
} catch (error) { if (error instanceof z.ZodError) { return res.status(401).json({  
message: 'Token inválido' }); } if (error instanceof jwt.JsonWebTokenError) { return  
res.status(401).json({ message: 'Token inválido ou expirado' }); } return  
res.status(500).json({ message: 'Erro interno do servidor' }); }
```

🔧 O que está acontecendo aqui?

- Se houver um erro na validação com `Zod` , **retorna um erro 401** :

json



Copy



Edit

```
{ "message": "Token inválido" }
```

- Se o token for **inválido ou expirado**, retorna:

json



Copy



Edit

```
{ "message": "Token inválido ou expirado" }
```

- Para **qualquer outro erro inesperado**, retorna:

json



Copy



Edit

```
{ "message": "Erro interno do servidor" }
```

Como Usar esse Middleware?

No Express, podemos usá-lo para proteger rotas privadas:

typescript



Copy



Edit

```
import express from 'express'; import { authMiddleware } from
'./middlewares/authMiddleware'; const app = express(); app.get('/perfil', authMiddleware,
(req, res) => { res.json({ message: `Bem-vindo, ${req.user.nome}!` }); });
```

Testando a API

✓ Requisição válida:

http



Copy



Edit

GET /perfil Authorization: Bearer eyJhbGciOi...

➡ Resposta esperada (200 OK):

json



Copy



Edit

```
{ "message": "Bem-vindo, João!" }
```

✗ Erro: Token ausente:

http



Copy



Edit

GET /perfil

➡ Resposta esperada (401 Unauthorized):

json



Copy



Edit

```
{ "message": "Token não fornecido" }
```

✗ Erro: Token inválido:

http

GET /perfil Authorization: Bearer token-falso



Copy



Edit

→ Resposta esperada (401 Unauthorized):

json



Copy



Edit

```
{ "message": "Token inválido" }
```

✗ Erro: Token expirado:

Se o JWT estiver expirado:

json



Copy



Edit

```
{ "message": "Token inválido ou expirado" }
```

🔥 Melhorias possíveis

1. Criar um tipo para `req.user`

- Atualmente, `req.user` pode ser `any`. Melhor definir um tipo:

typescript



Copy



Edit

```
declare module 'express' { export interface Request { user?: { id: number; nome: string }; } }
```

2. Suporte a múltiplos métodos de autenticação

- Atualmente, só aceita `"Bearer "`. Podemos permitir outros tipos de autenticação.

3. Customizar a expiração do token

- O tempo de expiração pode ser definido dinamicamente.



Resumo

- ✓ Valida o token JWT no cabeçalho `Authorization` .
- ✓ Decodifica os dados do usuário e os anexa à requisição.
- ✓ Rejeita requisições sem token ou com token inválido.
- ✓ Usado para proteger rotas privadas na API.