



AULA 07

FASTAPI

MIDDLEWARES E AUTENTICAÇÃO

O QUE VEREMOS HOJE

- 01 MIDDLEWARES
- 02 TIPOS DE MIDDLEWARES
- 03 MIDDLEWARE DE AUTENTICAÇÃO
- 04 CRIAÇÃO DE TOKEN DE AUTENTICAÇÃO
- 05 VALIDAÇÃO DO TOKEN DE AUTENTICAÇÃO

MIDDLEWARES

Middleware são funções ou partes de código que atuam como uma camada intermediária entre a requisição do cliente e a lógica de negócio da aplicação.

Eles são responsáveis por processar e modificar os dados da requisição antes de serem enviados para a lógica principal da aplicação.



TIPOS DE MIDDLEWARES

Autenticação: Responsáveis por validar as credenciais do usuário e controlar o acesso aos recursos da aplicação.

Registro: Registra informações sobre cada requisição, como o método, URL acessada, tempo de resposta, etc., para fins de monitoramento e auditoria.

Cache: Armazena respostas de requisições para reduzir o tempo de resposta e a carga no servidor em chamadas repetidas.
Exemplo: cache de respostas de APIs.



MIDDLEWARE DE REGISTRO (LOG)

Este middleware registra cada requisição feita ao servidor, exibindo o método HTTP, a URL e o tempo de processamento.

Esse tipo de middleware é útil para monitoramento e auditoria, ajudando a identificar o desempenho de cada rota.



MIDDLEWARE DE LOG

Para criar um middleware no FastAPI, temos o decorador `@app.middleware('http')`.

Esse método é indica que o middleware deve interceptar todas as requisições HTTP feitas.

E assim como nas rotas, precisamos criar a função que vai ser executada quando o middleware for acionado.

```
1 import time
2 from fastapi import FastAPI, Request
3 from database import create_tables
4 from routes import users
5
6 app = FastAPI()
7
8 # Adicionando o middleware de log na aplicação
9 @app.middleware("http")
10 def log_requests(request: Request, call_next):
11     start_time = time.time() # Iniciando o contador de tempo
12     print(f"Recebendo requisição: {request.method} {request.url}")
13
14     # Executando a requisição e recebendo a resposta
15     response = call_next(request)
16
17     # Calculando o tempo de resposta
18     process_time = time.time() - start_time
19     print(f"Requisição finalizada em {process_time:.2f} segundos")
20
21     return response
```

MIDDLEWARE DE AUTENTICAÇÃO

O middleware de autenticação é uma camada que intercepta todas as requisições e valida se o usuário está autenticado antes que elas alcancem as rotas da aplicação. No caso do FastAPI, podemos implementar essa autenticação usando tokens **JWT (JSON Web Tokens)**.

Para isso, o middleware verifica a presença de um token JWT válido no cabeçalho da requisição. Se o token for válido, a requisição é encaminhada para a lógica principal. Caso contrário, o middleware interrompe a requisição e retorna um erro de autenticação.



JWT

JWT (JSON Web Token) é um padrão aberto para transmitir informações de forma segura entre duas partes, como cliente e servidor.

O JWT é amplamente usado para autenticação e autorização em aplicações web e APIs, pois oferece uma forma segura e compacta de enviar informações entre partes.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IldpbGxpYW0iLCJpYXQiOiE1MTYyMzkwMjIsImV4cCI6NzI1ODEyOTIwMCwidXNlci1yb2x1IjoicmVndWxhciIsInVzZXItaWQiOiI0N0EzOEY2Mi05MzU3LTU4Q0YtMDRCNy0zNkNFMEM5Q0E2REYifQ.lzQu_eVt2Lh_1m412Dg9b_YxNKMp1RDnXFVfKykuk8
```


CONFIGURAÇÃO DO JWT

Para usar o JWT precisamos inicialmente instalar uma biblioteca que vai ajudar na criação e verificação dos tokens JWT.

Para instalar vamos rodar o comando:

pip install pyjwt

```
• (venv) aline@aline-Aspire-A515-54:~/WorkArea/Infinity/api_users$ pip install pyjwt
Collecting pyjwt
  Using cached PyJWT-2.9.0-py3-none-any.whl (22 kB)
Installing collected packages: pyjwt
Successfully installed pyjwt-2.9.0

[notice] A new release of pip available: 22.3.1 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
○ (venv) aline@aline-Aspire-A515-54:~/WorkArea/Infinity/api_users$
```

CRIAÇÃO DO TOKEN DE AUTENTICAÇÃO

Para confirmar o JWT, vamos precisar criar uma chave secreta. Essa chave é a “senha mestra” usada como assinatura de cada tokens.

Além disso, precisamos definir o método de criptografia para assinar o token. O HS256 é um dos algoritmos mais comuns e seguros para esse tipo de autenticação.

```
1 import jwt
2
3 # Configuração do segredo e do algoritmo de criptografia
4 SECRET_KEY = "chave_secreta" # A chave pode ser qualquer uma que você quiser
5 ALGORITHM = "HS256"
6
7 # Função para criar um token JWT
8 def create_token(dados: dict):
9     token = jwt.encode(dados, SECRET_KEY, ALGORITHM)
10    return token
```

CRIAÇÃO DO TOKEN DE AUTENTICAÇÃO

Uma vez tendo a função de criação do token, podemos usar onde quisermos. Nesse exemplo, a criação do token está sendo feita junto com a criação do usuário.

```
1 from fastapi import APIRouter, Depends, HTTPException
2 from sqlmodel import Session, select
3 from database import get_session
4 from models.user import User
5 from schemas.user import UserCreate
6 from middlewares.authentication import create_token
7
8 router = APIRouter()
9
10 @router.post("/users")
11 def post_users(user: UserCreate, session: Session = Depends(get_session)):
12     # Cria uma instância do usuário no modelo User com os dados recebidos
13     user_db = User(**user.model_dump())
14     print(user_db)
15
16     # Gera o token JWT para o novo usuário
17     token = create_token(dict(user)) # Função que gera o token usando os dados do usuário
18
19     # Armazena o token no campo 'token' do usuário
20     user_db.token = token
21
22     # Adiciona o novo usuário ao banco de dados
23     session.add(user_db)
24     session.commit()
25     session.refresh(user_db)
26
27     return {
28         "message": "User created",
29         "user": user_db,
30     }
```

ATIVIDADE PRÁTICA

No projeto da API de Registros de Chamados, crie uma rota que vai receber um email no corpo da requisição e vai gerar um token com base nesse email e a rota deve retornar o token gerado.

VALIDAÇÃO DO TOKEN

Para validar os tokens é necessário uma função que vai receber a requisição antes dela chegar na rota.


Essa função vai precisar buscar o token no header da requisição e verificar se é válido usando o `jwt.decode`.

```
1 import jwt
2 from fastapi import Request, HTTPException
3
4 # Configuração do segredo e do algoritmo de criptografia
5 SECRET_KEY = "chave_secreta" # A chave pode ser qualquer uma que você quiser
6 ALGORITHM = "HS256"
7
8 def auth_middleware(request: Request, call_next):
9     # Pega o token diretamente do cabeçalho Authorization
10    token = request.headers.get("Authorization")
11
12    # Verifica se o token está presente
13    if not token:
14        raise HTTPException(status_code=401, detail="Token not found")
15
16    # Decodifica o token JWT
17    payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
18
19    # Verifica se o payload é válido (não é None ou vazio, por exemplo)
20    if payload:
21        # Se o token for válido, chama a próxima etapa no ciclo de requisição
22        return call_next(request)
23    else:
24        # Caso o token seja inválido
25        raise HTTPException(status_code=401, detail="Invalid token")
```

VALIDAÇÃO DO TOKEN

Uma vez criado o middleware de autenticação, é necessário adicioná-lo ao aplicativo FastAPI para proteger as rotas.

A função `authenticate_request` chama o `auth_middleware`, que verifica se um token JWT válido foi incluído na requisição antes que ela chegue à rota desejada.

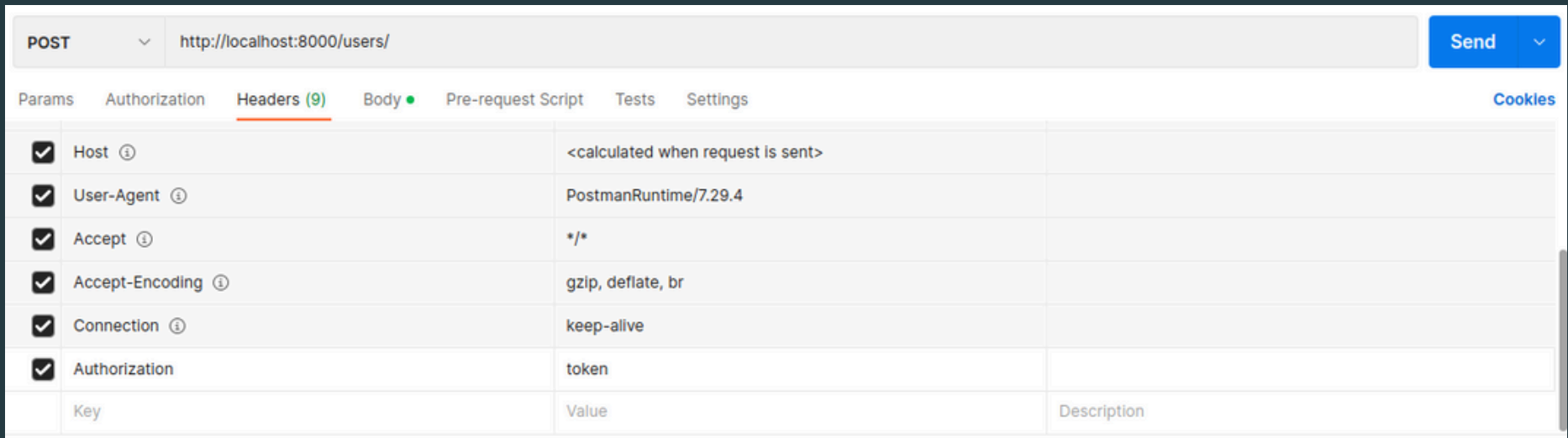


```
1  import time
2  from fastapi import FastAPI, Request
3  from database import create_tables
4  from routes import users
5  from middlewares.authentication import auth_middleware
6
7  app = FastAPI()
8
9  @app.middleware("http")
10 def authenticate_request(request: Request, call_next):
11     return auth_middleware(request, call_next)
```

REQUISIÇÃO COM TOKEN

Uma vez que foi adicionado o middleware de autenticação, precisamos fazer a requisição incluindo o token fornecido pela API.

No Postman isso pode ser feito na aba de Headers adicionando o Authorization e o token.



ATIVIDADE PRÁTICA

No projeto da API de Registros de Chamados, implemente o middleware de autenticação, só permitindo o acesso as rotas com o token.

O middleware deve retornar o status code 401 caso token seja inválido.