4. Autenticação JWT

O JWT (JSON Web Token) é utilizado para autenticar usuários e autorizar o acesso a rotas protegidas.

No k6, o fluxo de autenticação geralmente envolve duas etapas:

1. Fazer login para obter o token.
2. Utilizar esse token em requisições subsequentes.

Exemplo completo com login e uso do token

```
import http from 'k6/http';
import { check } from 'k6';

export function setup() {
  const loginPayload = JSON.stringify({
    email: 'fulano@qa.com',
    password: 'teste',
  });

  const loginHeaders = { 'Content-Type': 'application/json' };

  const login = http.post('http://localhost:3000/login', loginPayload, { head
ers: loginHeaders });

  const token = JSON.parse(login.body).authorization;
  console.log(`Token obtido: ${token}`);
  return { token };
}

export default function (data) {
  const headers = {
    Authorization: data.token},
    'Content-Type': 'application/json',
  };

  const res = http.get('http://localhost:3000/produtos', { headers });
```

```
    check(res, { 'status é 200': (r) => r.status === 200 });
}
```

Explicação detalhada

setup()

- A função `setup()` é executada **antes do teste iniciar**, apenas uma vez.
- É utilizada para preparar dados, realizar login e gerar tokens de autenticação.
- Tudo o que for retornado por ela é recebido como parâmetro na função principal (`default`).

Login e token

- `http.post('/login', payload, { headers })`: envia o e-mail e senha para autenticação.
- O servidor responde com um objeto JSON contendo o campo `authorization` (token JWT).
- `JSON.parse(login.body).authorization`: transforma o corpo da resposta em objeto e acessa o token.
- Esse token é retornado para ser reutilizado no teste.

Uso do token

- Dentro da função principal (`default`), o token é recuperado por `data.token`.
- O cabeçalho `Authorization` recebe o formato:

  ```
  Authorization: <token>
  ```

- Esse cabeçalho deve ser enviado em todas as requisições que exigem autenticação.

Validação com `check`

O método `check()` é usado para validar respostas:

```
check(res, { 'status é 200': (r) => r.status === 200 });
```

- O primeiro parâmetro é a resposta (`res`).
- O segundo é um objeto com as condições a serem verificadas.
- Caso a condição falhe, o k6 registra a falha na saída do teste.

Exemplo de requisição autenticada real

```
const headers = {
  Authorization: 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...',
  'Content-Type': 'application/json',
};

const res = http.get('http://localhost:3000/produtos', { headers });
```

---

5. Boas práticas

1. Centralize URLs e tokens em variáveis ou arquivos de configuração.
2. Use o `setup()` sempre que precisar gerar dados ou tokens antes dos testes.
3. Evite logs extensos (`console.log(res.body)`) em testes de carga, pois impactam o desempenho.
4. Utilize o `check()` para validar respostas críticas (status, campos obrigatórios, mensagens de erro).
5. Prefira simular cenários reais, por exemplo:
    - Login → criar produto → buscar produto → excluir produto.

---

6. Exemplo completo de fluxo com autenticação e CRUD

```
import http from 'k6/http';
import { check } from 'k6';

export function setup() {
```

```
  const loginPayload = JSON.stringify({
    "email": "fulano@qa.com",
    "password": "teste",
  });

  const loginHeaders = { 'Content-Type': 'application/json' };
  const loginRes = http.post('http://localhost:3000/login', loginPayload, { h
eaders: loginHeaders });
  check(loginRes, {
    'login bem sucedido': (r) => r.status === 200,
  });

  const authToken = loginRes.json('authorization');
  console.log(`Token de autenticação: ${authToken}`);
  return { authToken, baseUrl: 'http://localhost:3000' };
}

export default function (data) {
  const protectedHeaders = {
    'Content-Type': 'application/json',
    'Authorization': data.authToken,
  };

  const produto = http.post(`${data.baseUrl}/produtos`, JSON.stringify({
    nome: 'Produto Teste',
    preco: 120,
    descricao: 'Produto criado via teste automatizado',
    quantidade: 10,
  }), { headers: protectedHeaders });
  check(produto, {
    'produto criado com sucesso': (r) => r.status === 201,
  });
  const idProduto = produto.json('_id');

  const resGet = http.get(`${data.baseUrl}/produtos/${idProduto}`, { head
ers: protectedHeaders });
  check(resGet, {
    'consulta produtos': (r) => r.status === 200,
```

```
    });

    const resPut = http.put(`${data.baseUrl}/produtos/${idProduto}`, JSON.s
tringify({
        nome: 'Produto Teste',
        preco: 150,
        descricao: 'Produto criado via teste automatizado',
        quantidade: 10,
    }), { headers: protectedHeaders });
    check(resPut, {
        'produto atualizado': (r) => r.status === 200,
    });

}
```