Stress Test (Teste de Estresse)

Critérios de Aceitação

Objetivo

Validar a estabilidade, desempenho e tolerância a falhas do Serverest sob **carga progressiva crescente**, até atingir sobrecarga máxima e depois reduzir para observar recuperação.

Cenário – Fluxo de Produtos

Critérios de aceitação

- **Listagem de produtos** deve retornar **status 200**.
- O campo `produtos` deve ser um **array válido**.
- **Detalhe do produto** deve retornar **status 200**.
- O detalhe deve conter o campo **nome**.
- **Taxa de falhas** deve ser **menor que 5%** (`http_req_failed < 0.05`).
- **Pelo menos 90% dos checks** devem ser bem-sucedidos (`checks > 0.90`).
- **95% das requisições** devem responder em **menos de 2s** (`p(95) < 2000ms`).
- Deve executar com **ramping-vus**:
  - Começar com 10 **VUs**
  - Aumentar para **50 VUs em 1 minuto**
  - Atingir **700 VUs em 2 minutos**
  - Reduzir para **0 VUs em 3 minutos**

Cenário – Fluxo de Usuários

Critérios de aceitação

- **Criação de usuário** deve retornar **status 201**.

- **Consulta de detalhes do usuário** deve retornar **status 200**.
- **Taxa de falhas** deve ser **menor que 5%** (`http_req_failed < 0.05`).
- **Pelo menos 90% dos checks** devem ser bem-sucedidos (`checks > 0.90`).
- **95% das requisições** devem responder em **menos de 2s** (`p(95) < 2000ms`).
- Deve executar com **ramping-vus**:
  - Começar com **5 VUs**
  - Aumentar para **5 VUs em 1 minuto**
  - Atingir **100 VUs em 2 minutos**
  - Reduzir para **0 VUs em 3 minutos**

Script Stress Test Serverest (K6)

```
import http from 'k6/http';
import { check, group, sleep } from 'k6';

const BASE_URL = 'http://localhost:3000';

export const options = {
   scenarios: {
      produtos: {
         executor: 'ramping-vus',
         startVUs: 10,
         stages: [
            { duration: '1m', target: 50 },
            { duration: '2m', target: 700},
            { duration: '3m', target: 0 },
         ],
         exec: 'fluxoProdutos',
      },
      usuarios: {
         executor: 'ramping-vus',
         startVUs: 5,
         stages: [
            { duration: '1m', target: 5},
```

```
            { duration: '2m', target: 100 },
            { duration: '3m', target: 0 },
        ],
        exec: 'fluxoUsuarios',
    },
  },

  thresholds: {
    http_req_failed: ['rate<0.05'],  // até 5% de falhas toleradas
    checks: ['rate>0.90'],           // pelo menos 90% de checks bem-sucedi
dos
    http_req_duration: ['p(95)<2000'], // 95% das requisições < 2s
  },
};

export function fluxoProdutos() {
  group('Fluxo de Produtos', () => {
    const resListar = http.get(`${BASE_URL}/produtos`);
    check(resListar, {
      'Listar Produtos - status 200': (r) => r.status === 200,
      'Listar Produtos - array valido': (r) => Array.isArray(r.json('produto
s')),
    });

    const produtos = resListar.json('produtos');
    if (!produtos || produtos.length === 0) return;

    const index = Math.floor(Math.random() * produtos.length);
    const id = produtos[index]._id;

    const resDetalhe = http.get(`${BASE_URL}/produtos/${id}`);
    check(resDetalhe, {
      'Detalhe Produto - status 200': (r) => r.status === 200,
      'Detalhe Produto - nome existe': (r) => r.json('nome') !== undefined,
    });

    sleep(0.5);
  });
```

```
}

export function fluxoUsuarios() {
  group('Fluxo de Usuários', () => {
    const payload = JSON.stringify({
      nome: `Usuario${__VU}-${Date.now()}`,
      email: `usuario${__VU}-${Date.now()}@teste.com`,
      password: 'senha123',
      administrador: 'true',
    });

    const resCriar = http.post(`${BASE_URL}/usuarios`, payload, {
      headers: { 'Content-Type': 'application/json' },
    });

    check(resCriar, {
      'Criar Usuário - status 201': (r) => r.status === 201,
    });

    const id = resCriar.json('_id');
    if (!id) return;

    const resDetalhe = http.get(`${BASE_URL}/usuarios/${id}`);
    check(resDetalhe, {
      'Detalhe Usuário - status 200': (r) => r.status === 200,
    });

    sleep(0.5);
  });
}
```