3. Explicação dos testes e comentários do código

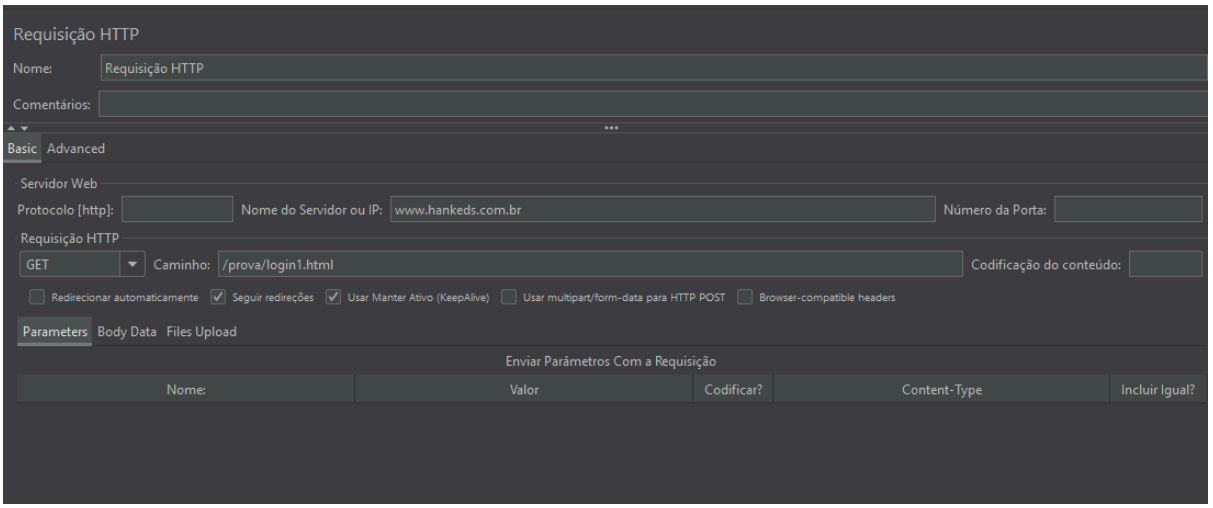
No teste com **Python**, usei o Selenium para abrir o navegador, acessar a página de login e preencher os campos de usuário e senha com o texto "admin" e "admin123456". Criei uma função que digita lentamente, simulando uma pessoa real digitando. Depois cliquei no botão de login e verifiquei se o endereço da página mudou para destino.html, o que indica que o login funcionou. Usei try e except para capturar erros e driver.quit() para fechar o navegador no fim.

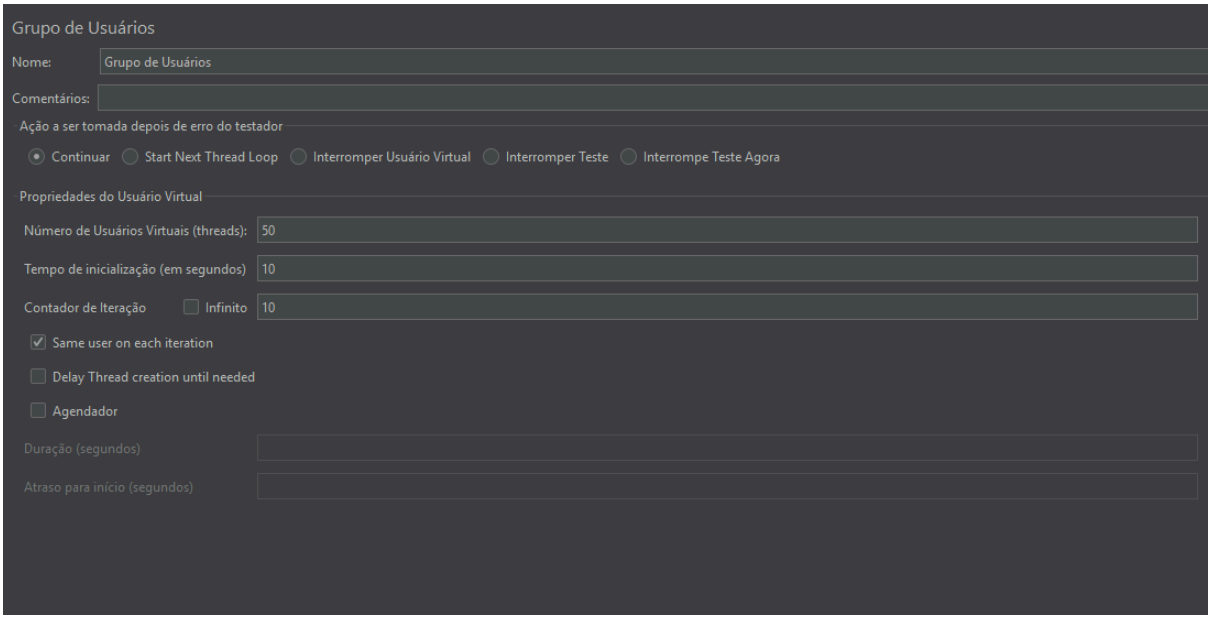
No teste com **Node.js**, a lógica foi a mesma: acessei a URL, esperei carregar, encontrei os campos e preenchi com a digitação letra por letra. Também usei uma pausa entre as teclas e depois cliquei no botão. No final, verifiquei se a URL tinha o texto esperado e fechei o navegador. O código foi feito de forma assíncrona com async/await, como é padrão no JavaScript.

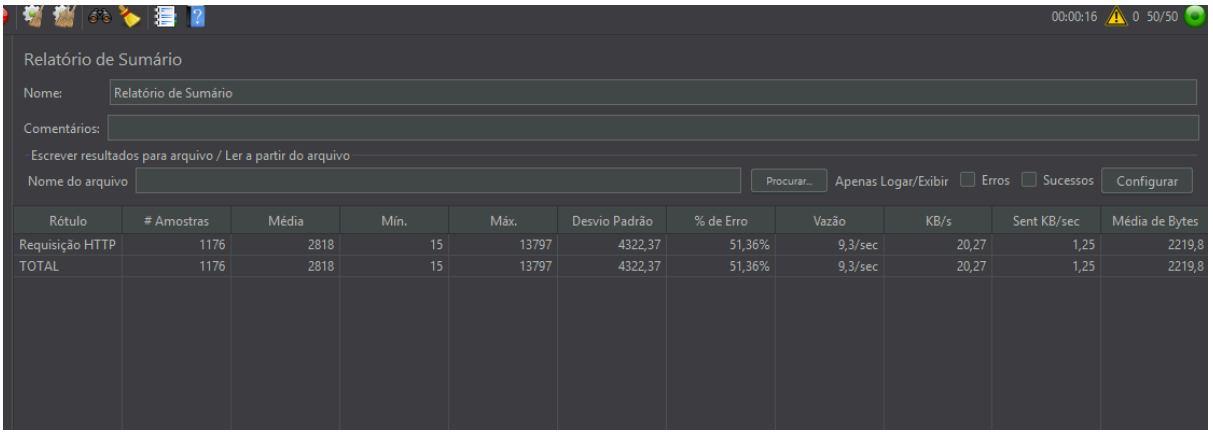
4. Como o Selenium funciona em Python e Node.js

O **Selenium** é uma ferramenta que automatiza o navegador, simulando ações de um usuário como clicar, digitar e navegar. Em **Python**, usamos webdriver e outras bibliotecas para esperar elementos da página e interagir com eles. Em **Node.js**, o funcionamento é parecido, mas usamos async/await para controlar o tempo de execução. Em ambas as linguagens, o Selenium executa testes de forma automática e eficiente, ideal para verificar se páginas estão funcionando corretamente.

5. JMeter







Análise dos testes de carga com JMeter

Foram realizados dois testes de desempenho nas páginas da aplicação, simulando diferentes quantidades de usuários e acessos, com o objetivo de identificar possíveis **problemas de performance ou instabilidade**.

Teste 1 – 70 usuários / 10 loops / ramp-up 10

Total de amostras: 27
Tempo médio de resposta: 9139ms (~9 segundos)
Erro: 18,52%
Tempo máximo: 17 segundos
Vazão (Throughput): 1,5 requisições por segundo
Mesmo com uma quantidade moderada de usuários, o tempo médio de resposta ficou bastante alto, ultrapassando 9 segundos. Isso indica que o servidor **demora para responder** mesmo sob uma carga relativamente leve. Além disso, quase **1 a cada 5 requisições falhou** (18,52%

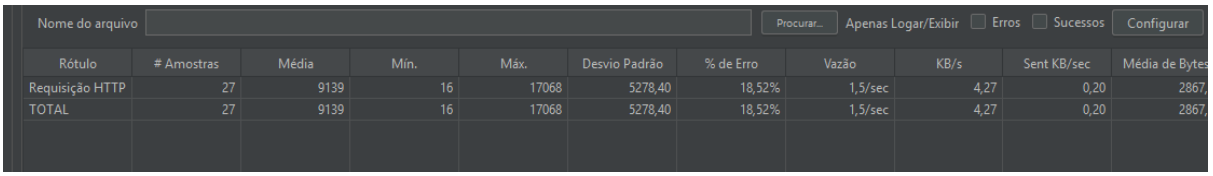
de erro), o que mostra que o sistema já começa a apresentar instabilidade com 70 usuários simultâneos.

Teste 2 – 150 usuários / 10 loops / ramp-up 10

Total de amostras: 336
Tempo médio de resposta: 32ms
Erro: 35,71%
Tempo máximo: 264ms
Vazão: 33,1 requisições por segundo

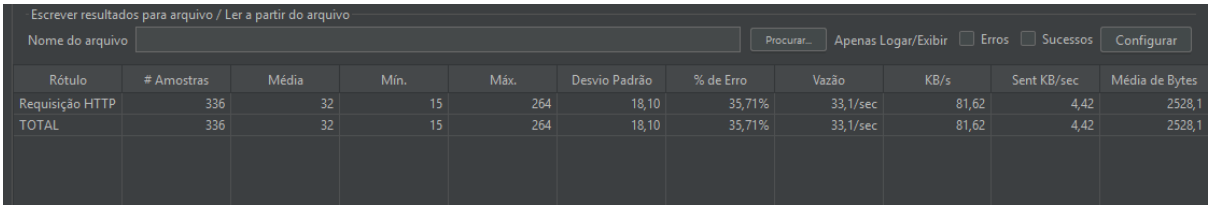
O tempo médio de resposta foi bom(32ms), mesmo com alta carga, mostrando que a aplicação está **otimizada para velocidade**. Porém, o número de erros subiu bastante, com **mais de 1/3 das requisições falhando** (35,71%). Isso indica que, embora o servidor seja rápido, **ele não está conseguindo lidar com o volume total de requisições** e começa a recusar ou falhar em algumas.

teste com 70/10/10



Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Requisição HTTP	27	9139	16	17068	5278,40	18,52%	1,5/sec	4,27	0,20	2867,
TOTAL	27	9139	16	17068	5278,40	18,52%	1,5/sec	4,27	0,20	2867,

teste com 150/10/10



Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Sent KB/sec	Média de Bytes
Requisição HTTP	336	32	15	264	18,10	35,71%	33,1/sec	81,62	4,42	2528,1
TOTAL	336	32	15	264	18,10	35,71%	33,1/sec	81,62	4,42	2528,1