Performance Testing Assessment – Restful Booker API **Date:** 4 September 2025

Contents

1. Introduction

Within this document I'll be outlining how I approached creating the performance testing assessment conducted against the RESTful Brooker API.

The objective of the assessment was to demonstrate proficiency in designing using JMeter to simulate and evaluate the performance of the Restful Booker API.

2. Test Approach

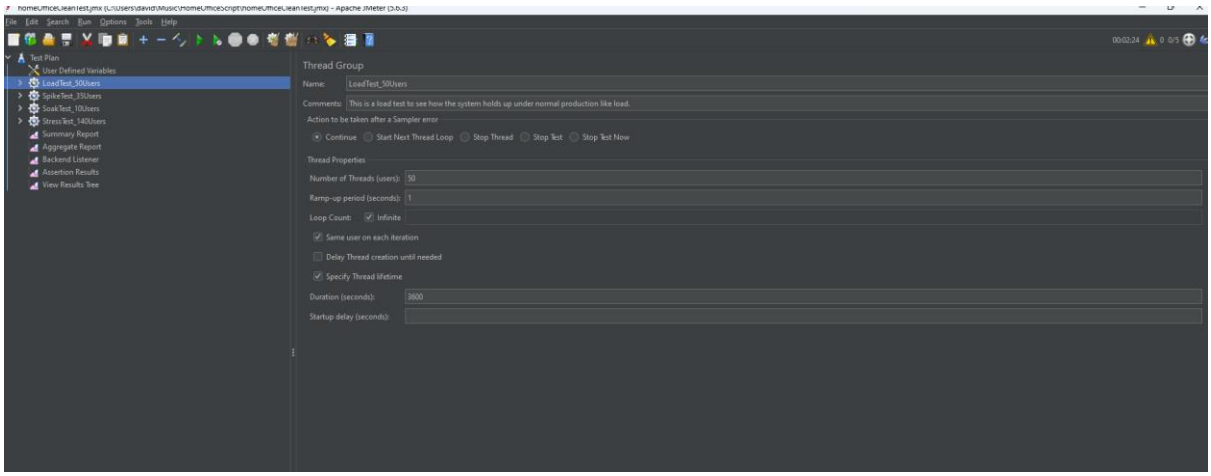
The test approach focused on covering all API key endpoints including Create, Get, Update, Patch, Delete and Auth. A modular script was created for maintainability. CSV Data Set Config and User Defined Variables were used for parameterisation. Dynamic values such as tokens and Booking IDs were extracted via JSN Extractors to ensure robustness.

Realistic user behaviour was simulated using Timers, and assertions were included to verify response codes and payload content. Several types of performances tests were implemented: Load, Stress, Soak and Spike each with appropriate thread groups and load profiles.

Only one of the thread groups was run at any kind of load and that was the Load Test. I ran this test for 15 minutes just to show that everything within the script is working as expected and I'll share the results below.

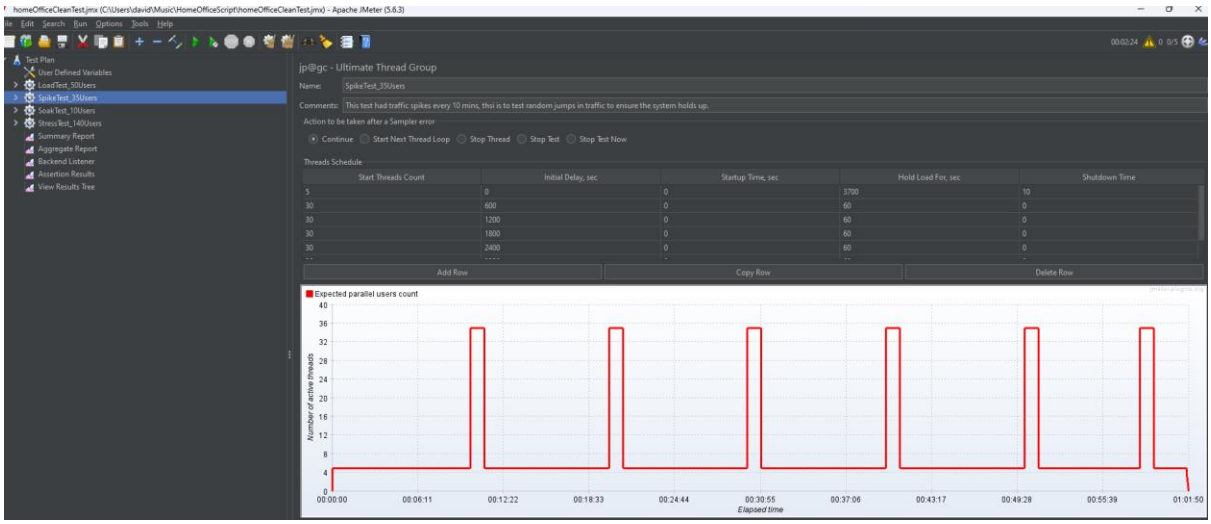
2.1 Load Test

The load test simulated 50 concurrent users performing realistic operations to determine the system's behaviour under expected peak load. This validated that the system could handle normal traffic levels without performance degradation.



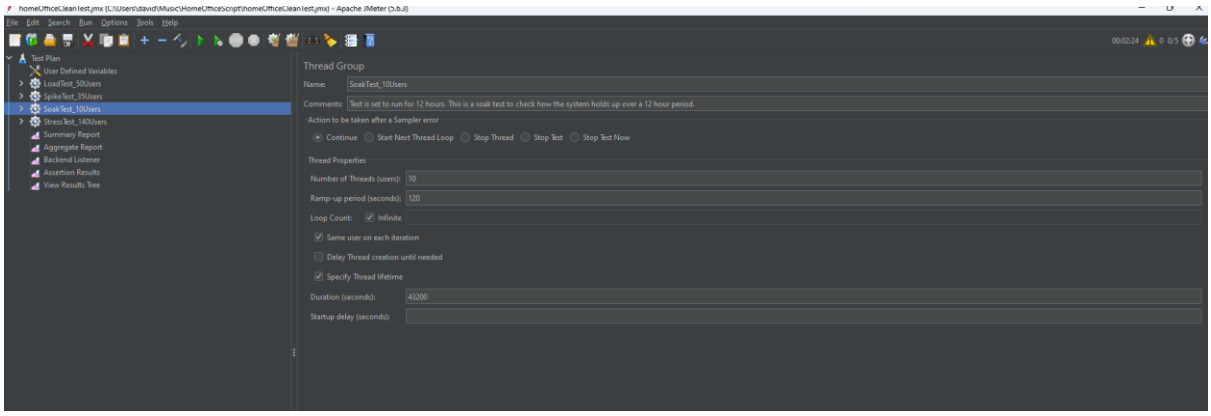
2.2 Spike Test

The spike test involved sudden surges in user load ramping from 5 to 30 users every 10 minutes, and the surge lasted for 1 minute, to observe how the system handled abrupt traffic increases. The goal here would be to detect any instability from unexpected spikes.



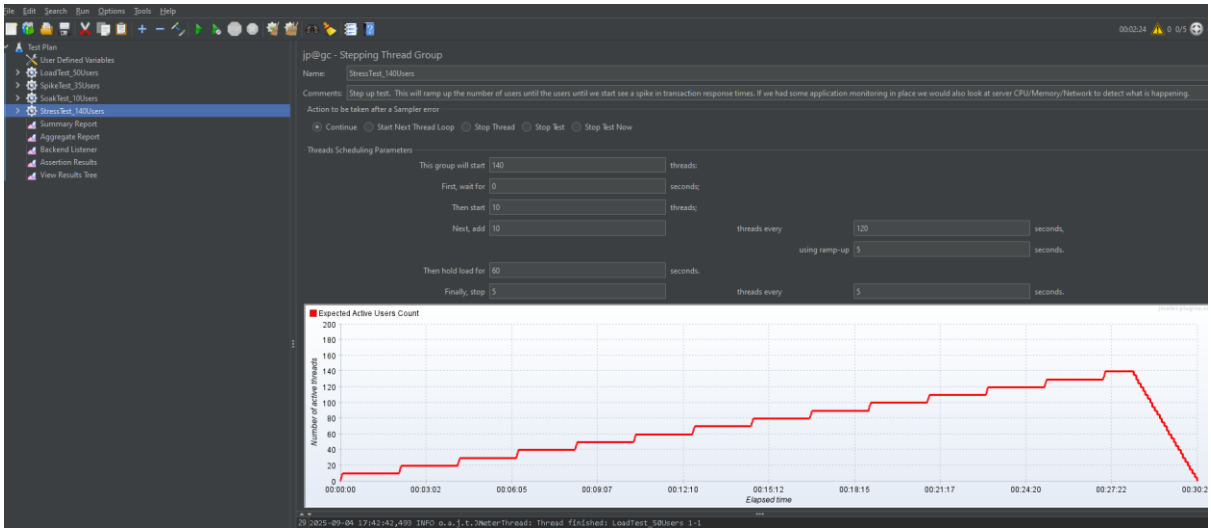
2.3 Soak Test

The soak test ran a moderate, steady load over an extended duration to identify issues like memory leaks or resource exhaustion. It helped ensure that the application remained stable and responsive over time. This test is a 12 hour test.



2.4 Stress Test

The stress test incrementally increased load beyond normal capacity until the system became unresponsive. This was used to identify breaking points, bottlenecks, and how the system recovered from failure.



3. JMeter Script Design

- **Parameterisation:** CSV Data Set Config (names, dates etc), User Defined Variables(admin/password, Host, Protocol), and Random String was used in different places to add in random strings.
- **Correlation:** Dynamic tokens and booking IDs were extracted using JSON extractors and reused in subsequent requests.
- **Assertions:** Status Codes and body content checks were used to ensure response validity.
- **Timers:** Timers were used to simulate real world behaviour.
- **Retry/Failure Logic:** Conditional retry logic was implemented for the Auth request using while Controllers and JSR223 Samplers.
- **Modularisation:** Each test type was grouped under separate Thread groups for clarity and control, also each transaction in each thread group was clearly named.
- **Endpoint Coverage:** Each endpoint was covered Create Auth/Get/Create/Updated/PartialUpdate/Delete BookingId.
-

4. Load Profile

As I don't have access to any production usage data (because it's a stub api, it was not possible to create a truly realistic load profile based on user behaviour. I made informed assumptions based on typical usage patterns for similar applications I have tested. Using these assumptions, I defined the estimated load profiles to cover various performance testing scenarios/types:

The profiles I designed are as follows:

- **Load Test**: 50 concurrent users over a 60 minute period (this is the only test actually executed and only executed for 15 minutes to demonstrate the test works).
- **Spike Test**: Starting with 5 users and spiking to 30 users for 1 minute every 10 minutes to simulate sudden bursts of traffic.
- **Stress Test**: Gradual ramp-up from 10 to 140 users to identify the system's break point.
- **Soak Test**: 10 users sustained over a 12 duration to evaluate system stability and resource usage over time.

5. Results Presentation

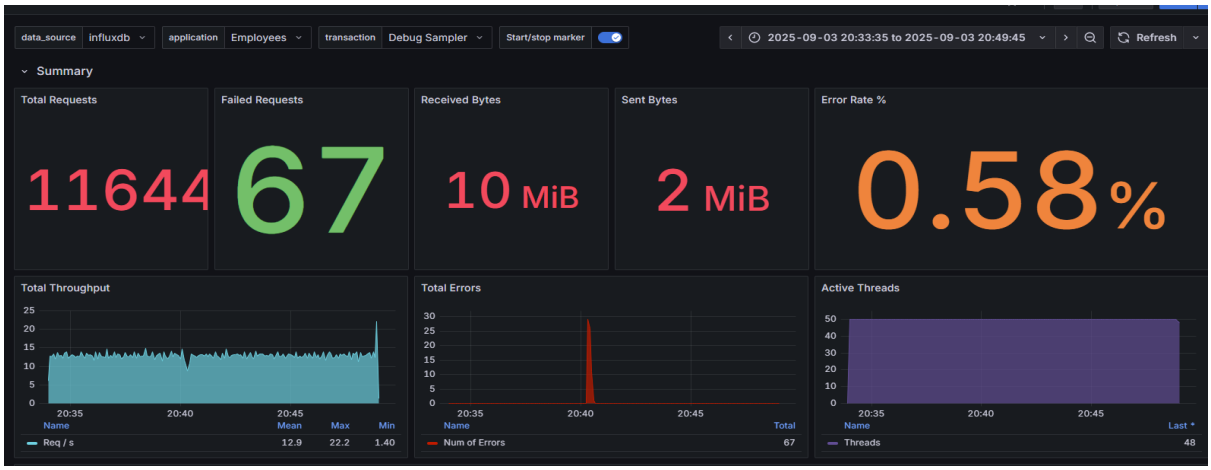
The results were captured using a Backend Listener linked to InfluxDB and visualised via Grafana dashboards. Key metrics including response time, error rate, and throughput were tracked.

Additional .jtl file was used to generate the report and the overview of the JMeter report will be included further down.

5.1 Grafana

Grafana was configured to provide real time visibility into system performance during the load test. With more time, additional graphs could be created and give better insights to what's happening during tests. This could be better statistics from transactions or even CPU/memory/network/disk from the server-side metrics.

- **Error Rate:** Peaked at 0.58%, indicating that the majority of requests were successful. 0.58% is well within acceptable error rate for most performance tests.
- **Throughput:** Maintained a consistent 12–13 requests/sec, suggesting stable system handling at load.
- **Active Threads:** Held steady at 50, confirming that user load was applied consistently throughout the test duration.
- **Errors:** There was a spike in errors around 20:40



5.2 JMeter Report

- Failure rate of 0.58% is looking good and within an acceptable threshold for load testing.
- 99 Percentile response times all looking good and below 450ms.
- Throughput was 13 transactions/sec, which indicates stable processing under sustained load.
- Overall the Average/90% look good.
- During the test there was some high Max times that could be investigated.

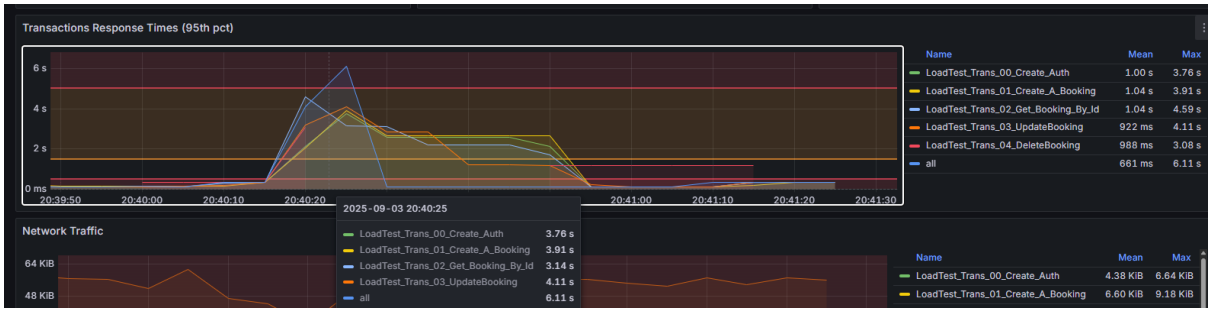
The system maintained a low error rate with consistent response times across all transactions. The average response time was comfortably under 150 ms, showing strong performance."

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**11647**	**67**	**0.58%**	**140.76**	**0**	**6407**	**110.00**	**129.00**	**327.00**	**430.52**	**12.98**	**11.05**	**4.57**
Create_Auth_Request	2693	1	0.04%	140.05	104	6107	108.00	137.60	327.00	443.00	3.00	2.26	0.91
CreateABooking	2682	3	0.11%	136.46	104	6407	109.00	124.70	325.00	339.85	3.00	2.77	1.38
GetBookingById	2667	22	0.82%	144.75	104	5117	110.00	127.00	326.60	367.56	3.00	2.68	0.68
LoadTest_Trans_00_Create_Auth	2704	1	0.04%	141.28	0	6107	108.00	137.50	327.00	443.00	3.00	2.25	0.90
LoadTest_Trans_01_Create_A_Booking	2692	3	0.11%	137.31	0	6407	109.00	124.70	325.00	344.00	3.00	2.76	1.37
LoadTest_Trans_02_Get_Booking_By_Id	2681	22	0.82%	143.99	0	5117	110.00	126.80	326.00	366.44	3.00	2.67	0.67
LoadTest_Trans_03_UpdateBooking	2667	36	1.35%	138.47	0	6122	112.00	128.00	327.00	341.00	3.00	2.66	1.38
LoadTest_Trans_04_DeleteBooking	900	5	0.56%	146.77	106	4111	112.00	148.00	330.00	358.99	1.02	0.75	0.25
Trans_03_UpdateBooking	2655	36	1.36%	137.99	106	6122	112.00	128.00	327.00	339.88	2.99	2.67	1.39
Trans_04_Delete_Booking	900	5	0.56%	146.77	106	4111	112.00	148.00	330.00	358.99	1.02	0.75	0.25

6. Bottlenecks and Recommendations

- **Auth:** There was failures sometimes for auth so I implanted the failure logic for that.
- **Spikes:** During the test there was some higher spikes around 20:40. If this was a real test we could investigate what happened here, what caused the spikes was it the CPU? Heap? Connection Pool? I'd recommend we check server logs and any potential APM tooling to see if we can get to the bottom of it.



7. Assessment Criteria Coverage

- **Robust Script Structure:** Separate thread groups for Load, Spike, Stress, and Soak; clear transactions and naming.
- **Correlation:** Dynamic values like bookingid and token were extracted and used in subsequent requests.
- **Realistic User Behaviour:** Timers like Gaussian Random timer were added to simulate think time. Also, a realistic load profile was used.
- **Endpoint Coverage:** All major end points were covered.
- **Success and Failure Validation**: Assertions were added for status codes and key response fields.
- **Data-Driven Testing:** CSV files were used to supply different usernames and booking data and User Defined Variables was used for different variables.
- **Test Types:** Load, Spike, Stress, and Soak test structures were created with clearly defined user models.
- **Failure Recovery:** Retry mechanism implemented for selected requests using JSR223 and counters.
- **Advanced Techniques/JMeter Usage:** Used Cookie Manager, Header Manager, Debug Sampler and used backend listener to implement InfluxDB/Grafana.
- **Report & Analysis:** Test results visualised via InfluxDB + Grafana, and metrics included in the summary report.