BlazeDemo Performance Test Plan

Test Design

The goal of this test is to simulate real users booking flights on BlazeDemo website.
We create 3 end-to-end journeys, each go through `reserve`, `purchase` and `confirmation` pages.
All requests use HTTP POST and we check the response and parse HTML to make sure correct result return.
There is also small random think time between each journey step to make traffic looks more real.

Each journey flow:

1. Search flight (`/reserve.php`)
2. Choose flight and purchase (`/purchase.php`)
3. Submit payment and verify receipt (`/confirmation.php`)

I use **k6** tool because it easy to control arrival rate and gives clear metrics like p(95) and error rate.

Monitoring

During run, we need monitor:

- `http_req_duration` p(99)
- Failure rate (`http_req_failed`)
- Custom metrics: `html_parse_failures`, `receipt_miss`, `booking_total_ms`
- System CPU and memory usage of target web server
- k6 output to IGrafana dashboard for live chart

Analysis

After test, I will compare:

- Response time distribution between 3 journeys
- Error pattern (most from confirmation step or reserve step)
- Average booking time vs expected SLA
- Resource utilization of backend

If some steps show p(99) > 1000 ms or high failed check, we will deep dive using server logs and network traces.

Points to Consider

We should put some APM or instrumentation inside the app, not only test from outside.
When we only use k6 or JMeter, we see latency and error, but don't know what layer make it slow.
With APM agent (for example New Relic, Datadog, Elastic or OpenTelemetry), we can check:

- which endpoint or controller is slow
- which SQL or external API make delay
- GC (if Java) pause or thread lock problem
- slow 3rd party call
  So having APM inside test env help find real backend problem much more fast than only black box test.

Appendix: Little's Law Explanation

For this requirement , I am using little's law

Put a total load of **600** concurrent users across 3 e2e journeys, with 50% load on Paris to London (Journey#1) bookings, and rest 2 journeys with 25% load each.

Sometimes we not control user by fixed number, but by how many request come per second.
Little's Law give the relation between **concurrent user**, **arrival rate**, and **average time**.

The formula is very simple:
L = λ × W

Where:

- **L** = average number of users in system (concurrency)
- **λ** = arrival rate (transactions per second)
- **W** = average time one journey take (seconds)

For example, if we want around **600 concurrent users**, and one full booking journey take around **2 second**,
then:
`λ = L / W = 600 / 2 = 300 request/sec.`

So we can setup total arrival rate to 300 rps, and it will make system have around 600 active users on

average.

Then we split the total rate by journey ratio 50% / 25% / 25%:

- Journey#1 (Paris → London): 150 rps
- Journey#2 (Mexico City → Berlin): 75 rps
- Journey#3 (Portland → Dublin): 75 rps

This way load is more realistic and still matching 600 users target.

Little's Law help us to plan load test better, not just guess number of VU, but calculate it from transaction time.

How to Use Little's Law in My Script

Step 1 – Get average journey time (W)

First I run small load test, just few requests, maybe 1-2 minutes, to check how long one full booking journey take from reserve to confirmation.

From the result I can see metric like `booking_total_ms` or I calculate average manually.

If average time is around 2.2 seconds, then **W = 2.2**.

This number is important, because it decide how many request we need per second.

Example command for small test:

```
k6 run -e RATE_J1=5 -e RATE_J2=3 -e RATE_J3=3 -e DURATION=1m -e RAMPUP=10s blazedemo_test.js
```

Step 2 – Calculate total arrival rate (λ)

If I want to simulate 600 users, I can use formula:

λ = L / W

So **λ = 600 / 2.2 ≈ 273 requests per second**.

Then split it by journey ratio (50%, 25%, 25%):

- Journey#1 → 136 rps
- Journey#2 → 68 rps
- Journey#3 → 68 rps

Then I run test like:

```
k6 run -e RATE_J1=136 -e RATE_J2=68 -e RATE_J3=68 -e DURATION=3m -e RAMPUP=30s blazedemo_test.js
```

This way total 300 rps will produce around 600 active users on average.