



HUSTEF

HUNGARIAN SOFTWARE TESTING FORUM

Load testing websites following the steps of a user journey

Raphael Roems

Contents

- Why is load testing important?
- What is a user journey?
- Why perform load testing following the steps of a user journey?
- Creating test scenarios and test data
- Load testing website backend and frontend architecture
- Test tool selection
- Environment(s) to use when executing load tests
- Test results
- Final thoughts

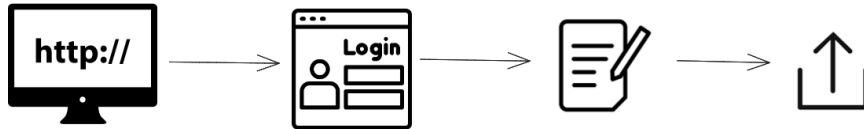
Why is load testing important?

- Assists in identifying performance related shortcomings in a software product
- Load test results being a deliverable for business stakeholders
- Preventing sub optimal user experience and potential revenue loss/extra costs
- Findings present an opportunity to fine-tune and improve the software

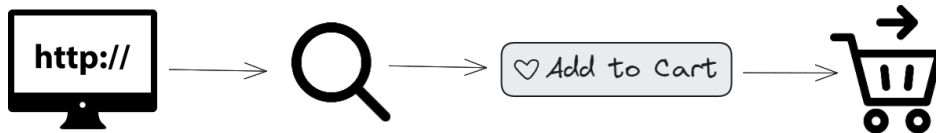
What is a user journey?

Specific steps a user needs to perform to accomplish something on a website

Educational website



Ecommerce website



Why perform load testing following the steps of a user journey?

1000s of users
only accessing
a homepage



1000s of users
simultaneously
browsing website
using most
common
functionality
with unique
test data

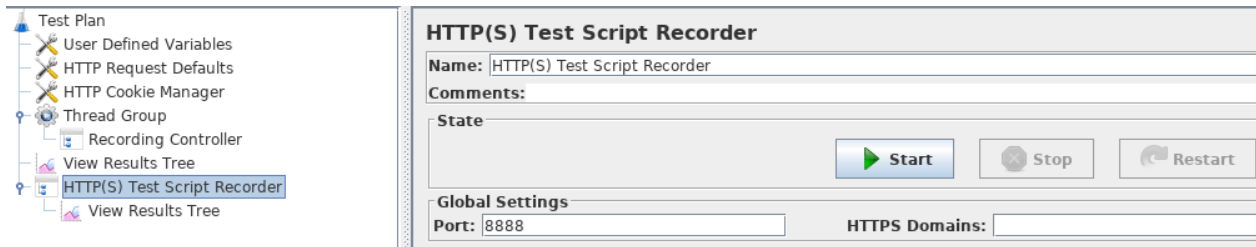
Why perform load testing following the steps of a user journey?

Identifying bottlenecks including:

- Building up of website objects (calculations)
- Third-party service providers (authentication services, payment gateways)
- High CPU/memory usage, database related issues (slow queries, locks)

Creating test scenarios

- Browse as a user would
- Focus on the most common use cases, rather than all the possible use cases
- Ensure that sufficient and valid test data is available for test duration
- Recording tools



The screenshot displays a test plan interface. On the left, a tree view shows the following structure:

- Test Plan
 - User Defined Variables
 - HTTP Request Defaults
 - HTTP Cookie Manager
 - Thread Group
 - Recording Controller
 - View Results Tree
 - HTTP(S) Test Script Recorder** (highlighted)
 - View Results Tree

The right pane shows the configuration for the selected **HTTP(S) Test Script Recorder**:

- Name:** HTTP(S) Test Script Recorder
- Comments:**
- State:** Includes buttons for **Start** (green play icon), **Stop** (grey stop icon), and **Restart** (grey refresh icon).
- Global Settings:**
 - Port:** 8888
 - HTTPS Domains:** (empty text field)

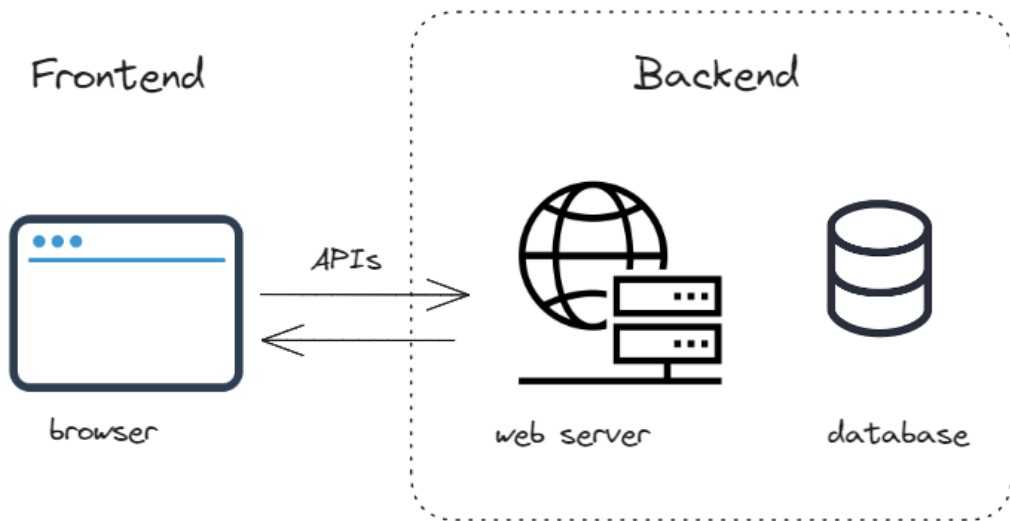
Using unique cookies for user journey tests

The screenshot shows a web browser window displaying the homepage of emag.hu. The page features a large blue banner with a percentage sign icon and the text "A nap ajánlata" (Today's offer). Below the banner, there are images of various household appliances and cleaning products, including a box of LAICA STREAM, a bottle of ARIEL detergent, and a container of Lenor fabric softener. A red callout box highlights a "20% kedvezmény" (20% discount) on selected household appliances. Another red callout box mentions "Különböző tisztítószer és mosószer csomagajánlatok" (Different detergent and fabric softener package offers). A blue button labeled "Megértettem" (I understand) is visible.

Below the browser window, the network inspector is open, showing a list of requests. The selected request is from www.emag.hu, and its details are displayed in the right pane. The cookies section shows a list of cookies, including a unique session cookie:

```
Cookie: EMAGVISITOR=a%3A1%3A%7B%3A7%3A%22user_id%22%3B%3A23230502850360067808%3B%7D; ituid=1692442685.440-fde860ad3c6bf22ad338d6f3dc5bfd4046a731b7; EMAGUID=1692442685-4637623881-76186.610; site_version_11=not_mobile; etc.
```


Load testing website backend and frontend architecture



Load testing backend performance

Backend

Frontend

Targets underlying application servers

API testing can target specific components

Less resource-intensive than frontend performance testing, more suitable for generating high load

User experience aspect is not factored in

Load testing frontend performance

Backend	Frontend
<p data-bbox="106 491 627 529">Targets underlying application servers</p> <p data-bbox="86 616 676 655">API testing can target specific components</p> <p data-bbox="77 726 763 797">Less resource-intensive than frontend performance testing, more suitable for generating high load</p> <p data-bbox="96 868 666 906">User experience aspect is not factored in</p>	<p data-bbox="830 507 1352 546">Verifies performance on interface level</p> <p data-bbox="830 606 1362 644">Concerned with the end-user experience</p> <p data-bbox="840 720 1429 791">Dependency on fully integrated environments and cost of scaling these are high</p>

Challenges with load testing frontend performance

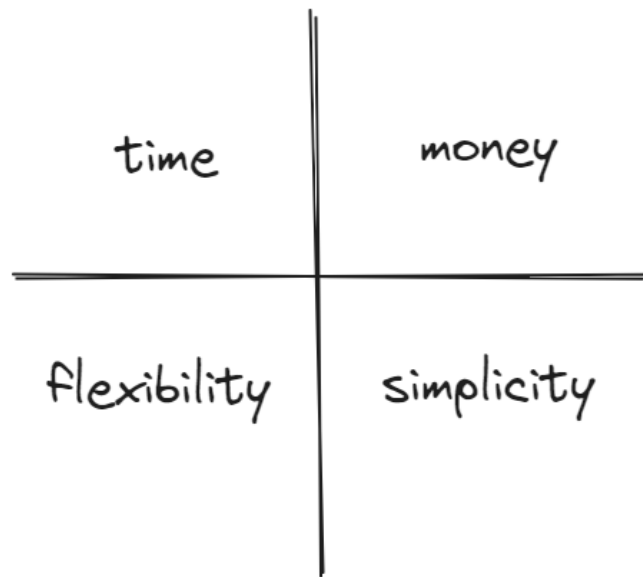
COMPARE PLANS

Month-to-Month Annual - Save up to 35%

Starter FREE	Basic \$99/month	MOST POPULAR \$5,000 80,000 VUH/Year ⓘ 2 Mock Services 5 Hour Max Test Duration	Unleashed Contact for Pricing
50 Concurrent Users 10 Tests 1 Mock Service 20 Minute Max Test Duration	1,000 Concurrent Users 200 Tests/Year 1 Mock Service 1 Hour Max Test Duration	Each Virtual User (Thread) in your test will consume 1 VUH for tests of 1-60 minutes, 2 VUH for tests of 61-120 minutes, etc... Each GUI Functional Test will consume 100 VUH	Volume Discounts Fixed Cost (Unlimited) Plans Test Data Dedicated IPs & On Premise Options Priority Support
SIGN UP FOR FREE	BUY NOW	BUY NOW	TALK TO US

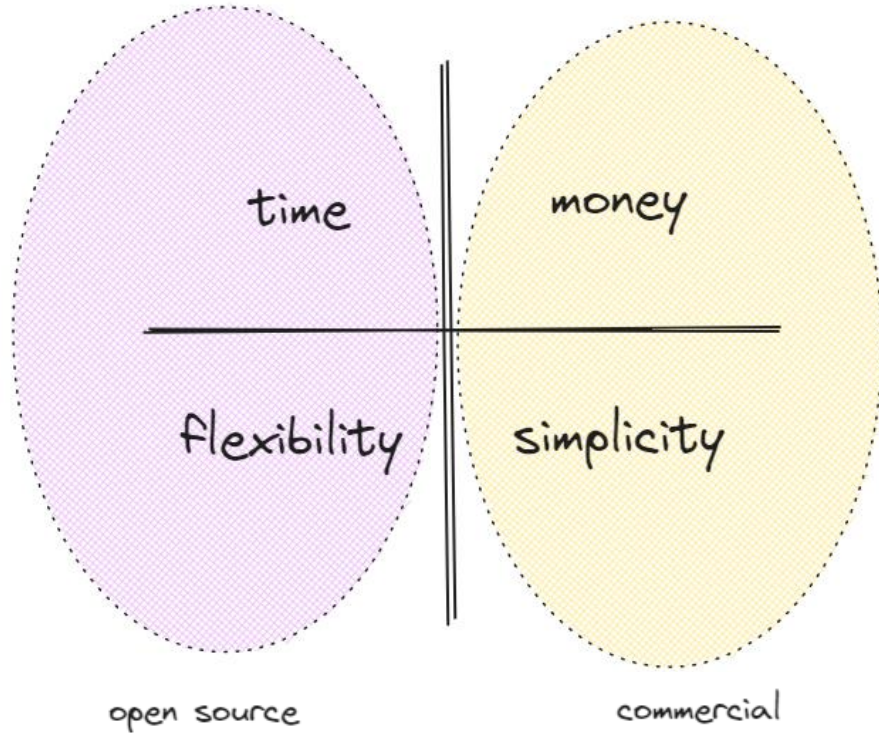
Source: Blazemeter

Test tool selection

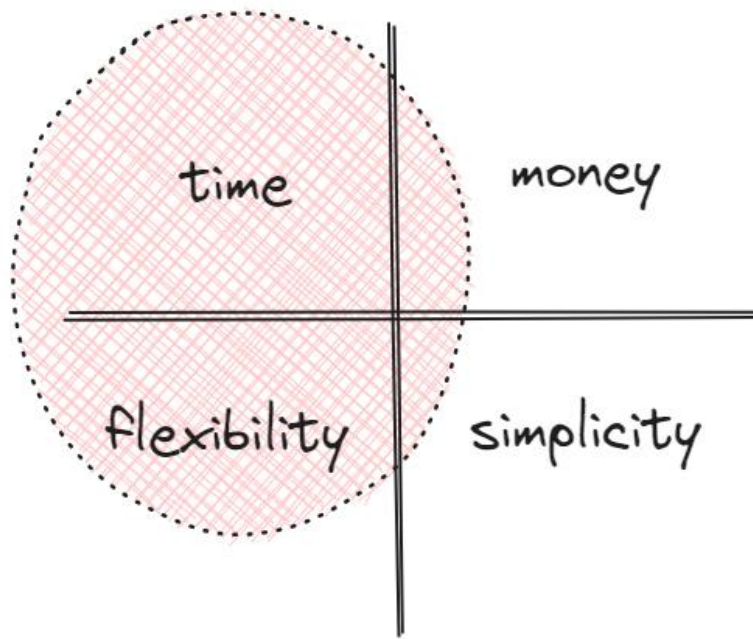
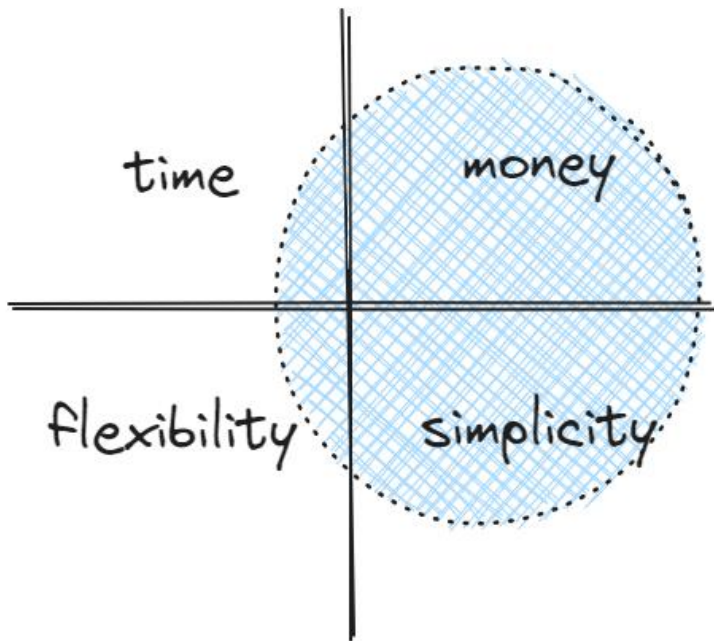


trade-offs between different tools

Test tool selection



Test tool selection

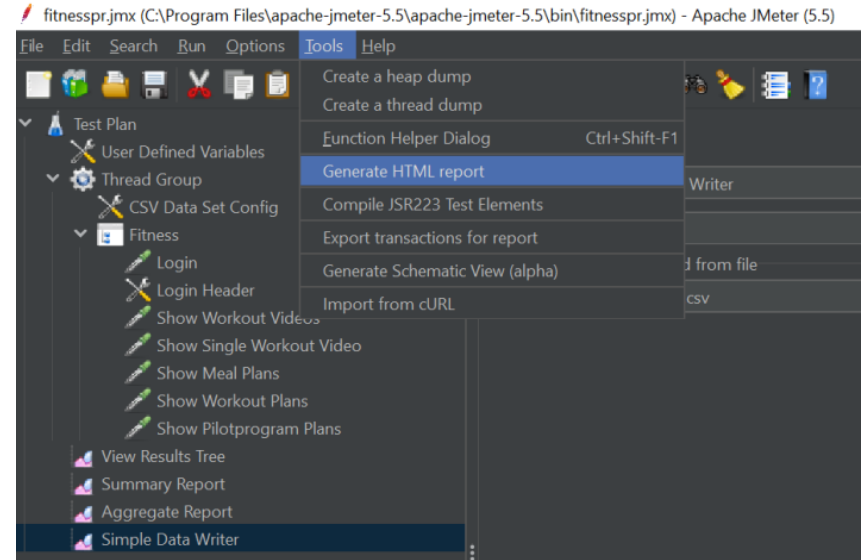
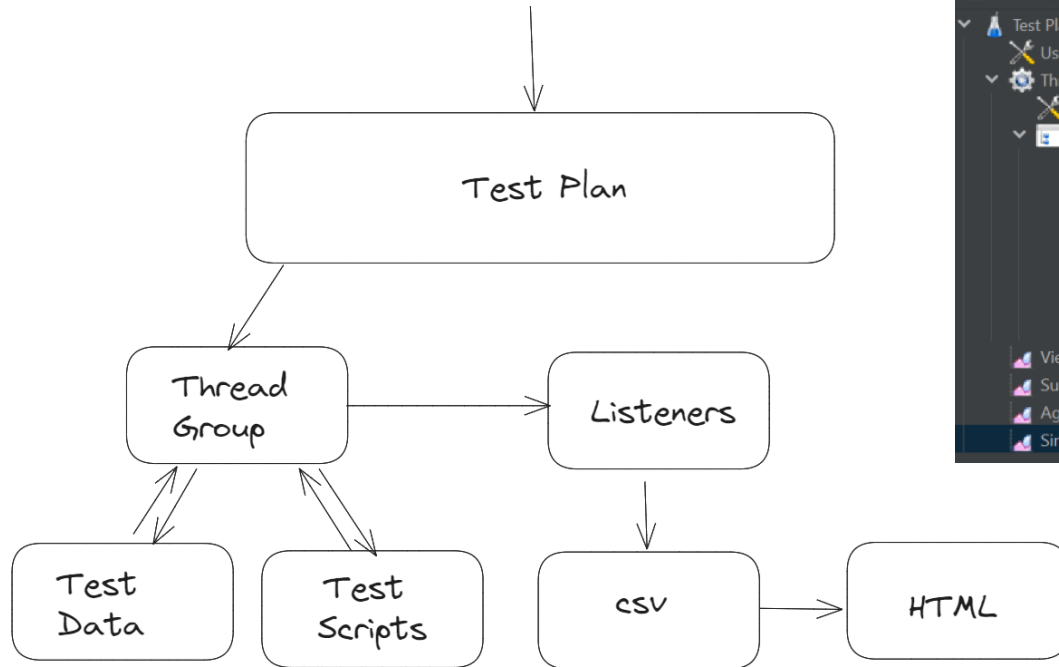


Environment(s) to use when executing load tests

- Environment used for load testing should ideally have same hardware and software configuration as production
- Testing in production yields most accurate results, but has risks!
- Testing in a pre-production environments enables the identification of performance related defects early
- Attempting to extrapolate load test results to different environment has risks

Test results (JMeter)

non GUI execution



Test results (JMeter)

SAMPLE LOAD TEST DASHBOARD

- Dashboard
- Charts
- Customs Graphs

Test and Report information

Source file	"Sample_100th0RP.csv"
Start Time	"4242"
End Time	"4245"
Filter for display	""

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.662	500 ms	1 sec 500 ms	Total
0.075	500 ms	1 sec 500 ms	GetCategory
0.910	500 ms	1 sec 500 ms	Category.Joke
1.000	500 ms	1 sec 500 ms	Random.Joke

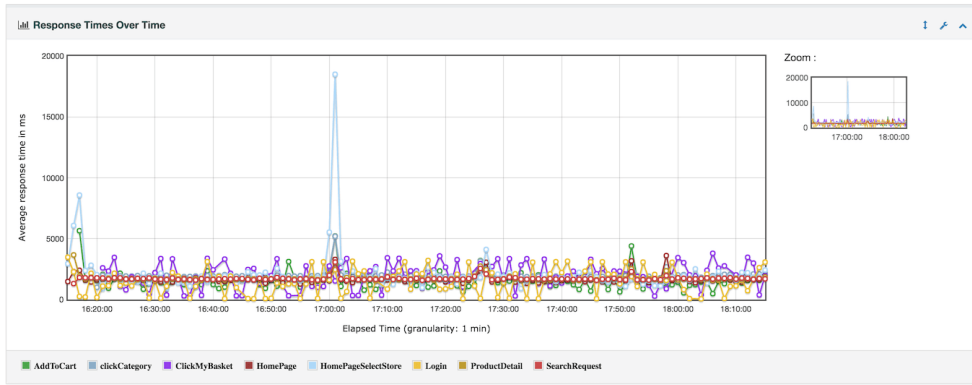
Requests Summary



Test results (JMeter)

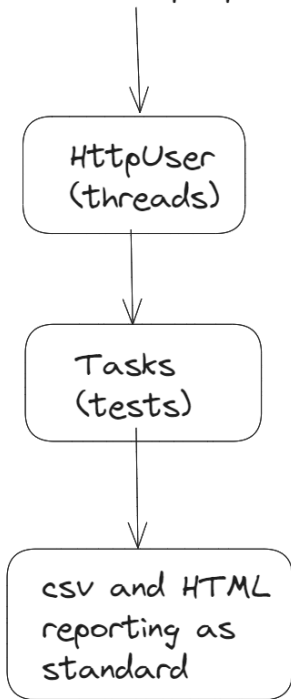
Statistics

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
Total	300	9	3.00%	716.33	107	2084	302.00	1759.00	1822.60	1914.97	117.14	118.92	17.45
CategoryJoke	100	9	9.00%	226.23	107	487	202.00	365.20	477.85	486.97	170.36	184.35	26.87
GetCategory	100	0	0.00%	1660.49	1335	2084	1663.00	1836.80	1875.05	2083.54	47.98	41.76	7.03
RandomJoke	100	0	0.00%	262.26	125	451	270.00	336.90	350.85	450.81	157.73	172.33	22.49



Test results (Locust)

no GUI, strictly Python scripting



```
import time
from locust import HttpUser, task, between

class QuickstartUser(HttpUser):
    wait_time = between(1, 5)

    @task
    def hello_world(self):
        self.client.get("/hello")
        self.client.get("/world")

    @task(3)
    def view_items(self):
        for item_id in range(10):
            self.client.get(f"/item?id={item_id}", name="/item")
            time.sleep(1)
```

Test results (Locust)

Locust Test Report

During: 2023/08/30, 14:06:40 - 2023/08/30, 14:17:05

Target Host: <https://service-testing.staging.dev/>

Script: locustfile.py

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/authorise	3372	154	5638	63	61117	595	5.4	0.2
POST	/add	1734	81	5767	94	60537	520	2.8	0.1
PUT	/update	1678	229	7368	90	60817	608	2.7	0.4
Aggregated		6784	464	6099	63	61117	579	10.9	0.7

Response Time Statistics

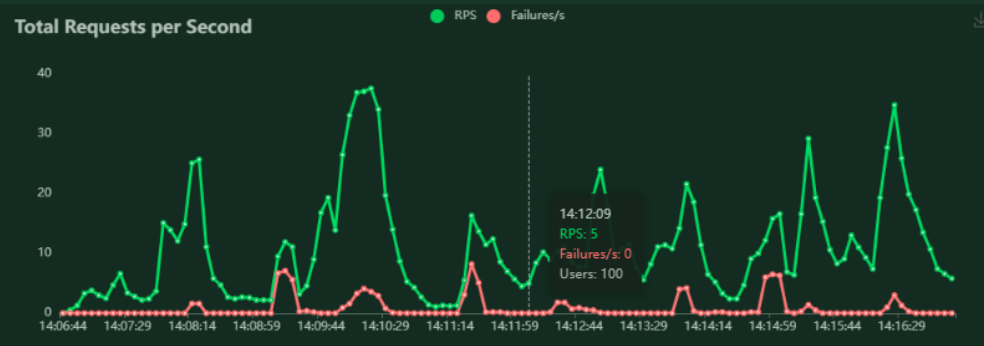
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/authorise	880	930	1000	3000	20000	40000	60000	61000
POST	/add	400	2100	3000	3800	19000	37000	60000	61000
PUT	/update	1100	1200	1300	4500	28000	58000	60000	61000
Aggregated		930	1000	1300	3600	21000	43000	60000	61000

Test results (Locust)

Failures Statistics

Method	Name	Error	Occurrences
PUT	/update	HTTPError(400 Client Error: Bad Request for url: https://service-testing.staging.dev/update')	153
POST	/add	HTTPError(504 Server Error: Gateway Time-out for url: https://service-testing.staging.dev/add')	22
POST	/authorise	HTTPError(504 Server Error: Gateway Time-out for url: https://service-testing.staging.dev/authorise')	36
POST	/authorise	HTTPError(502 Server Error: Bad Gateway for url: https://service-testing.staging.dev/authorise')	118
PUT	/update	HTTPError(504 Server Error: Gateway Time-out for url: https://service-testing.staging.dev/update')	74
POST	/add	HTTPError(502 Server Error: Bad Gateway for url: https://service-testing.staging.dev/add')	59
PUT	/update	HTTPError(502 Server Error: Bad Gateway for url: https://service-testing.staging.dev/update')	2

Charts



Test results (Locust)



Tasks

Ratio per User class

- 100.0% QuickstartUser
 - 25.0% add_emails
 - 25.0% updates
 - 50.0% login

Total ratio

- 100.0% QuickstartUser
 - 25.0% add_emails
 - 25.0% updates
 - 50.0% login

Final thoughts

- Proper preparation before starting load testing will prevent issues down the line
- JMeter is the preferable tool to use when starting an internal load/performance testing competency
- When there is a skill or time sensitive load testing requirement, a popular commercial tool can be useful
- Locust has distinct advantages when there is sufficient technical expertise

Questions?