DP2 – PERFORMANCE

GI - 01

Diāna Bukša

Manuel Cañizares Juan

Yoana Dimitrova Penkova

Iván Menacho Gallardo

Álvaro Rubia Tapia

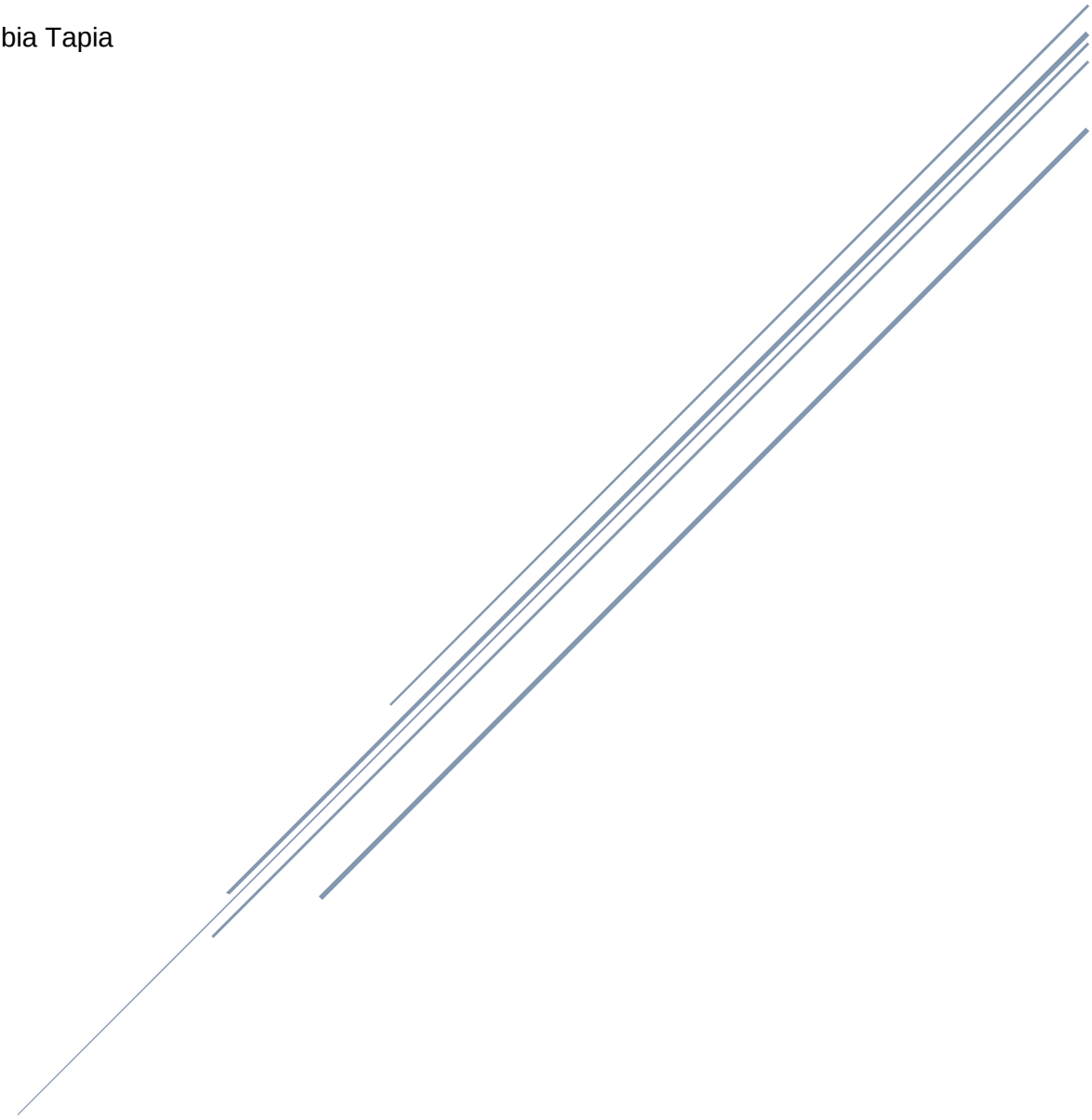


Table of Contents

Introduction

In order to develop this document, we decided to divide all user stories accordingly to which ones we develop, we studied the use of Gatling and we take some considerations in order to be consistent with our results:

- A stress test will be considered successful if it gets more than a 90% of errors. This threshold may very depend of computer limitations.
- A load test will be considered successful if it gets between 2000ms and 1000ms mean response

Each member of the group did the report individually, so there is some variation in the focus each one of us took.

Individual Reports

Álvaro

For the purpose of this deliverable, I have been in charge for the user stories I implemented in the application for the previous springs.

US (id)	Name
US-10	User Adopts a Pet
US-14	Vet schedules new Intervention
US-15	Vet plans Intervention
US-16	Owners sees Interventions
US-17	Owners sees Vet’s personal information

The 10th and 15th user stories are create feature whereas the 14th,16th and 17th are listing feature. Between those who has the same feature the results are very similar, and the number use to test the performance are the same. This decision of using the same number had in mind save time due to lack a capable computer where to run tests fast.

I follow the instructions in the video provided in the EV. First, I started working on us-10 and us-15, hen move to the others.

The first test was the more challenging because I was not sure of the number of users I should start with. Once I knew the intervals, I had to try each us was easier than the last one, having to judge just by the complexity of the feature.

PC specs

Aspire E1-522, age 8 years.

Processor

- CPU Type A4
- Processor Number A4-5000
- Manufacturer AMD
- Clock Speed 1.5 GHz

Ram

- Technology DDR3L SDRAM
- Installed Size 4 GB

Memory

- Max Supported Size 8 GB
- Technology DDR3L SDRAM

Performance Analysis

US-10

- Load Test

-1500 users. While running test with more than 1500 users, some request started failing/been rejected/taking a lot of time. (I tested adding or subtracting +-200 users per test)

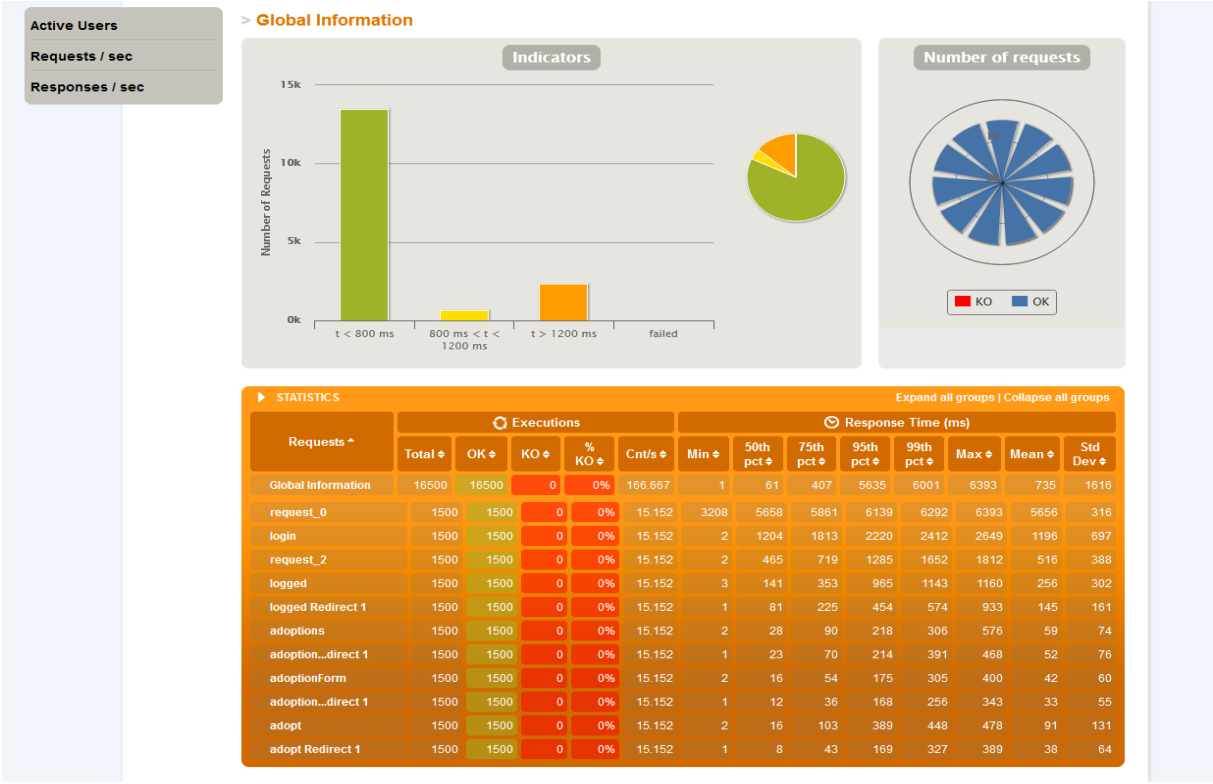


Figure 1 US-010 Load test

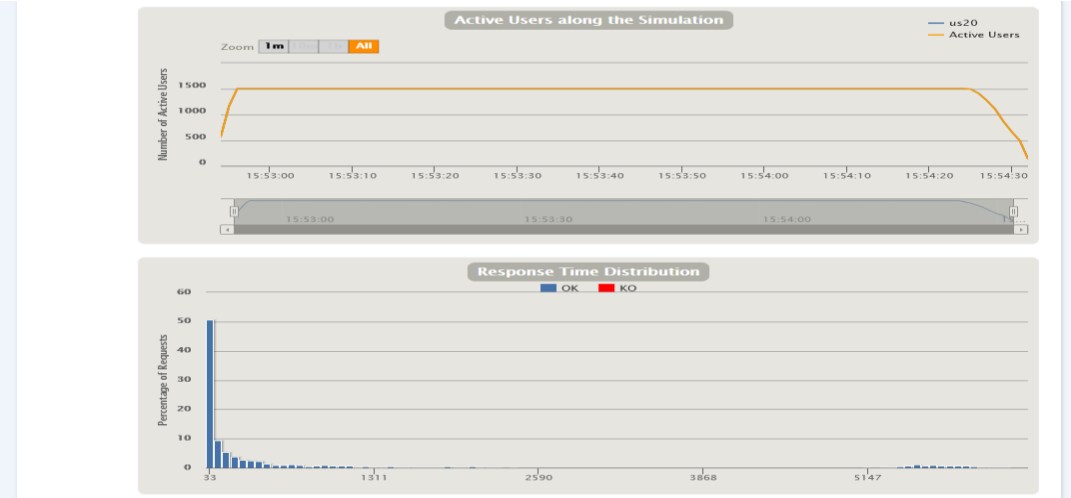


Figure 2 US-010 Load test graph

- Stress test

The failure point for this user story has been 70000 current users.

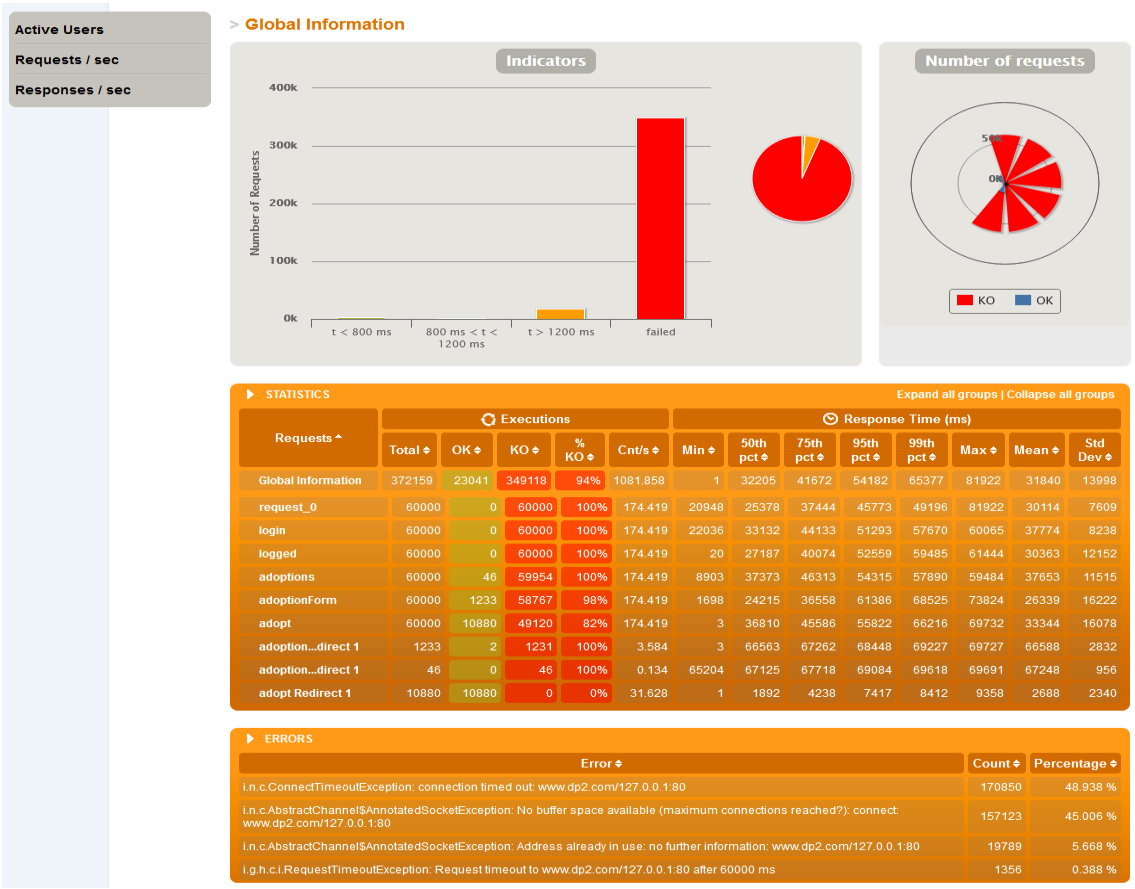


Figure 3 US-011 Stress test

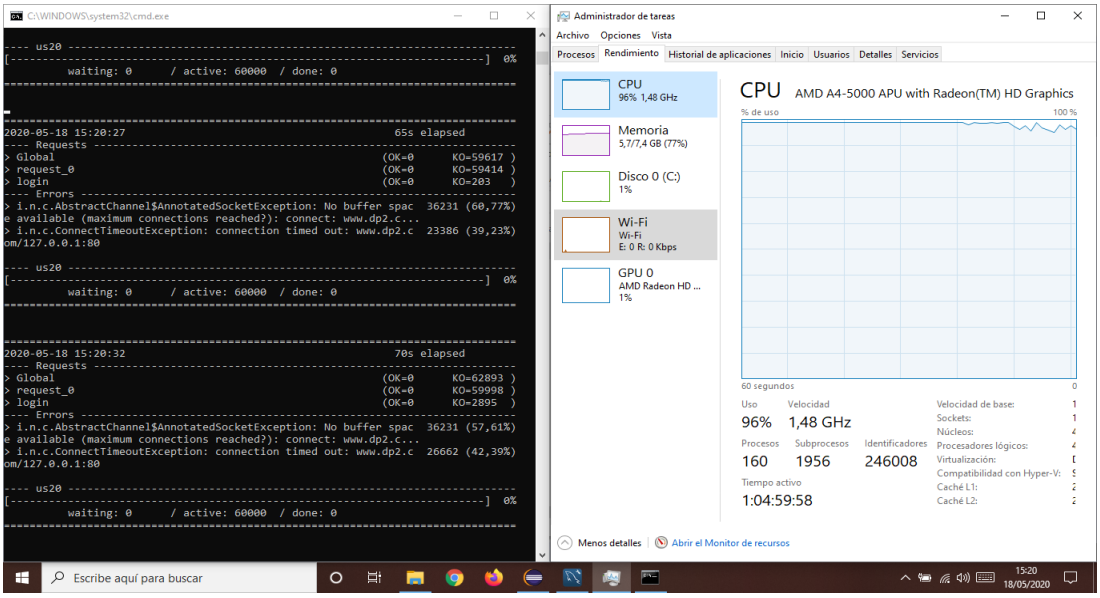
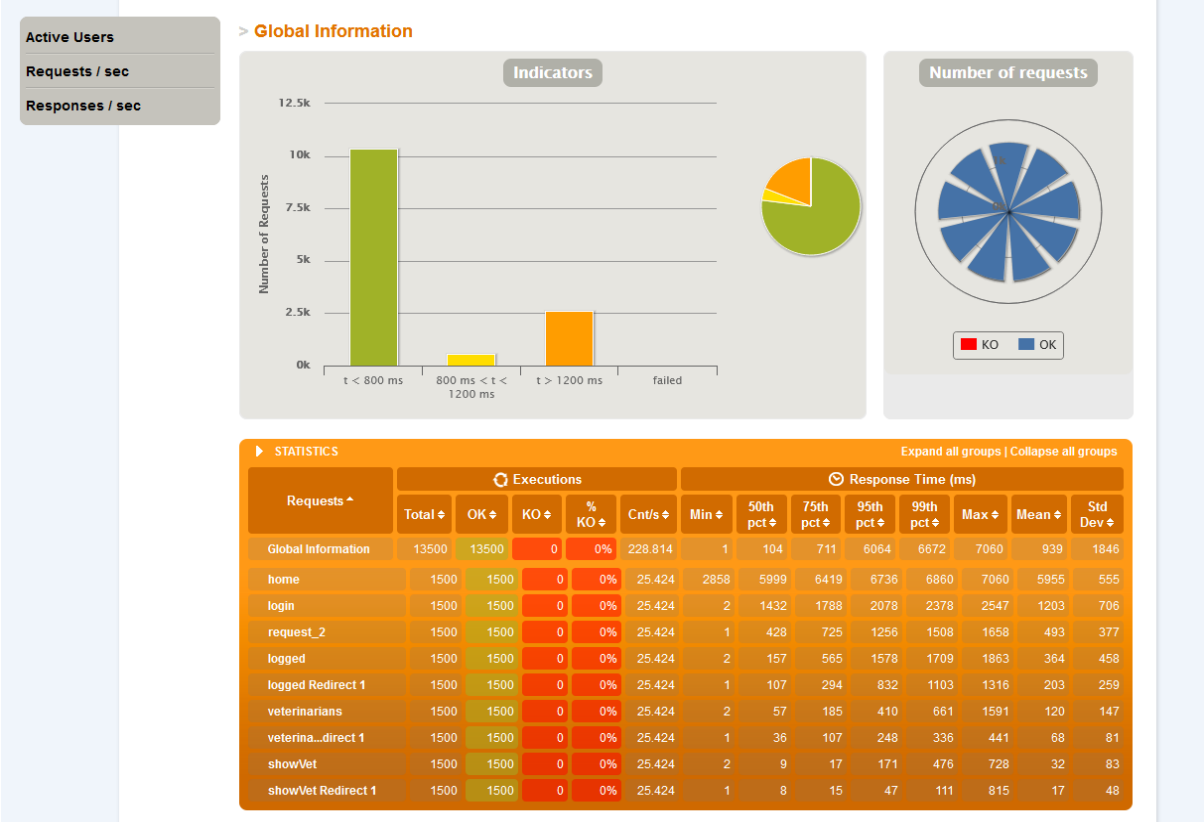


Figure 4 US-011 Stress test performance

US-14

- Load test

-1500 users. While running test with more than 1500 users, some request started failing/been rejected/taking a lot of time. (I tested adding or subtracting +-200 users per test)



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	13500	13500	0	0%	228.814	1	104	711	6064	6672	7060	939	1846
home	1500	1500	0	0%	25.424	2858	5999	6419	6736	6860	7060	5955	555
login	1500	1500	0	0%	25.424	2	1432	1788	2078	2378	2547	1203	706
request_2	1500	1500	0	0%	25.424	1	428	725	1256	1508	1658	493	377
logged	1500	1500	0	0%	25.424	2	157	565	1578	1709	1863	364	458
logged Redirect 1	1500	1500	0	0%	25.424	1	107	294	832	1103	1316	203	259
veterinarians	1500	1500	0	0%	25.424	2	57	185	410	661	1591	120	147
veterina...direct 1	1500	1500	0	0%	25.424	1	36	107	248	336	441	68	81
showVet	1500	1500	0	0%	25.424	2	9	17	171	476	728	32	83
showVet Redirect 1	1500	1500	0	0%	25.424	1	8	15	47	111	815	17	48

Figure 5 US-014 Load test

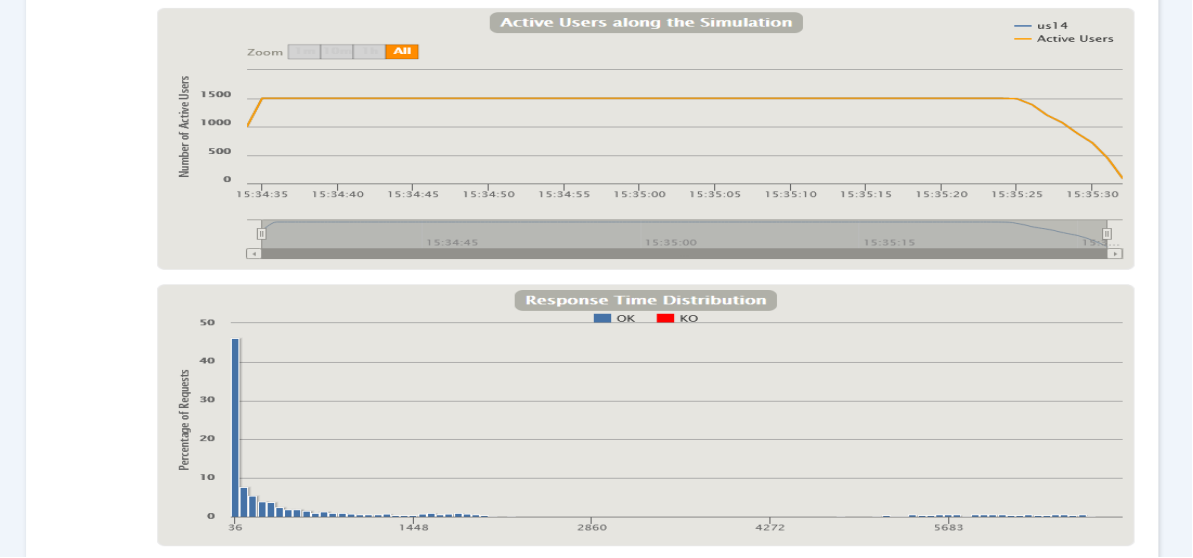


Figure 6 US-014 Load test graph

- Stress test

The failure point for this user story has been 65000 current users.

Figure 7 US-014 Stress test

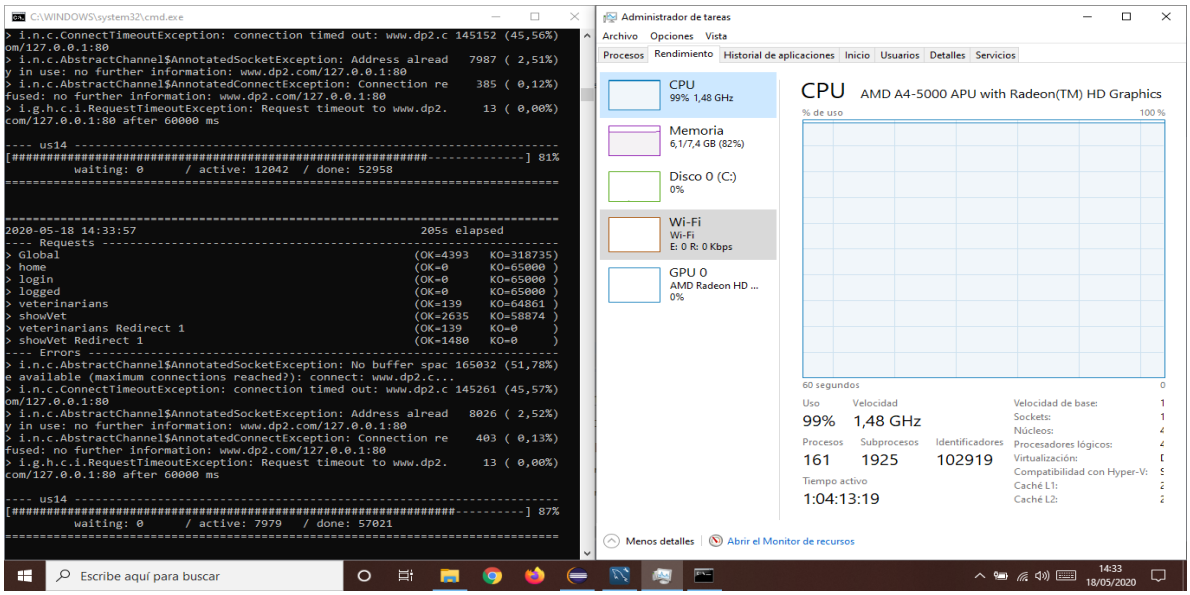
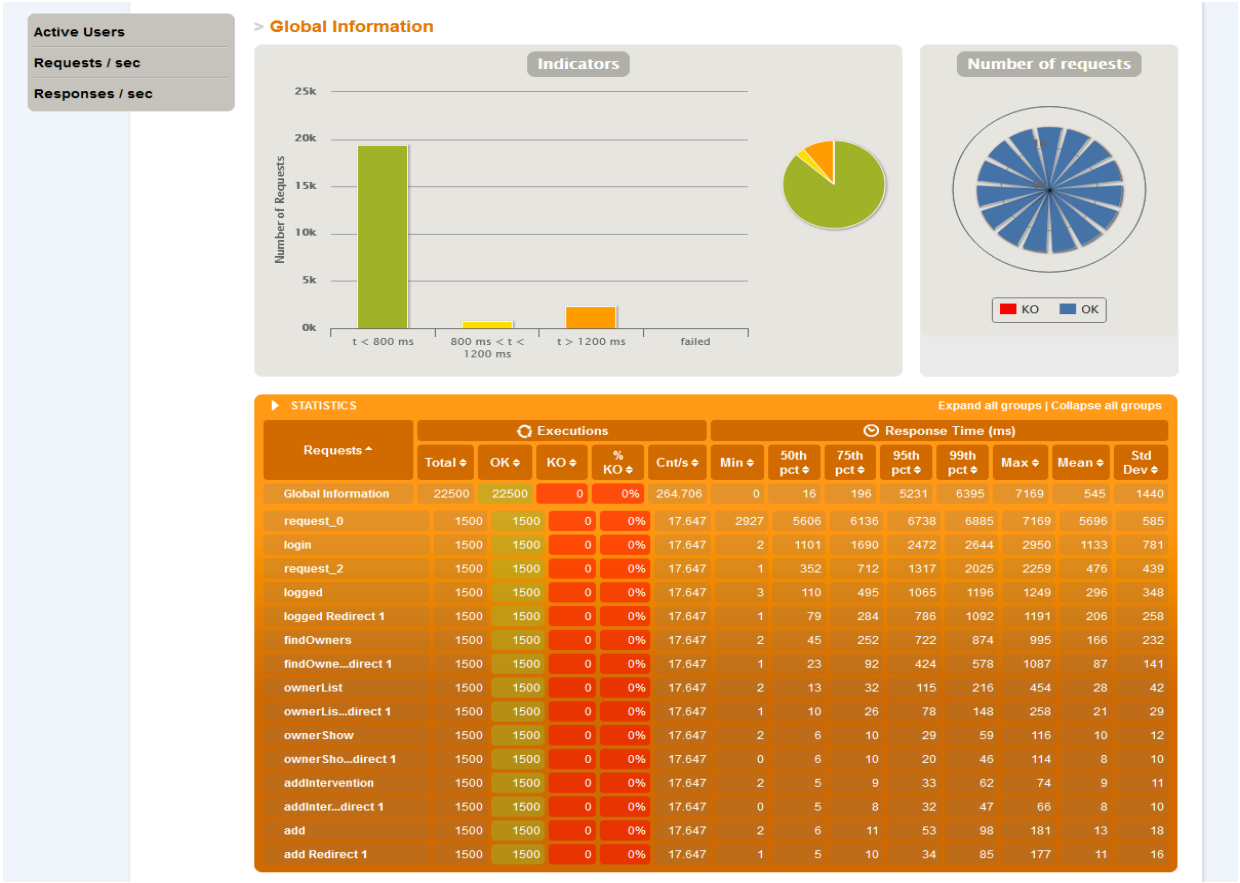


Figure 8 US-011 Stress test performance

US-15

- Load test

-1500 users. While running test with more than 1500 users, some request started failing/been rejected/taking a lot of time. (I tested adding or subtracting +-200 users per test)



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	22500	22500	0	0%	264.706	0	16	196	5231	6395	7169	545	1440
request_0	1500	1500	0	0%	17.647	2927	5606	6136	6738	6885	7169	5696	585
login	1500	1500	0	0%	17.647	2	1101	1690	2472	2644	2950	1133	781
request_2	1500	1500	0	0%	17.647	1	352	712	1317	2025	2259	476	439
logged	1500	1500	0	0%	17.647	3	110	495	1065	1196	1249	296	348
logged Redirect 1	1500	1500	0	0%	17.647	1	79	284	786	1092	1191	206	258
findOwners	1500	1500	0	0%	17.647	2	45	252	722	874	995	166	232
findOwne...direct 1	1500	1500	0	0%	17.647	1	23	92	424	578	1087	87	141
ownerList	1500	1500	0	0%	17.647	2	13	32	115	216	454	28	42
ownerLis...direct 1	1500	1500	0	0%	17.647	1	10	26	78	148	258	21	29
ownerShow	1500	1500	0	0%	17.647	2	6	10	29	59	116	10	12
ownerSho...direct 1	1500	1500	0	0%	17.647	0	6	10	20	46	114	8	10
addIntervention	1500	1500	0	0%	17.647	2	5	9	33	62	74	9	11
addInter...direct 1	1500	1500	0	0%	17.647	0	5	8	32	47	66	8	10
add	1500	1500	0	0%	17.647	2	6	11	53	98	181	13	18
add Redirect 1	1500	1500	0	0%	17.647	1	5	10	34	85	177	11	16

Figure 9 US-015 Load test

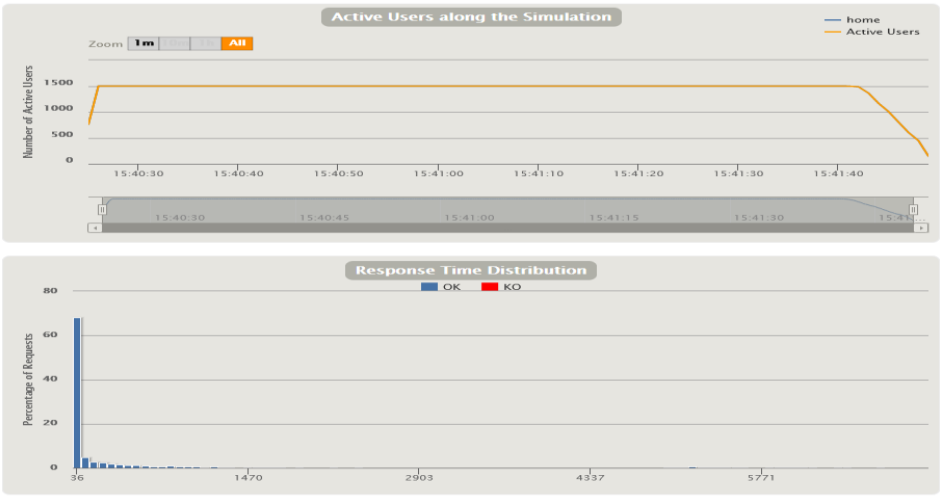


Figure 10 US-015 Load test graph

- Stress test

The failure point for this user story has been 50000 current users.

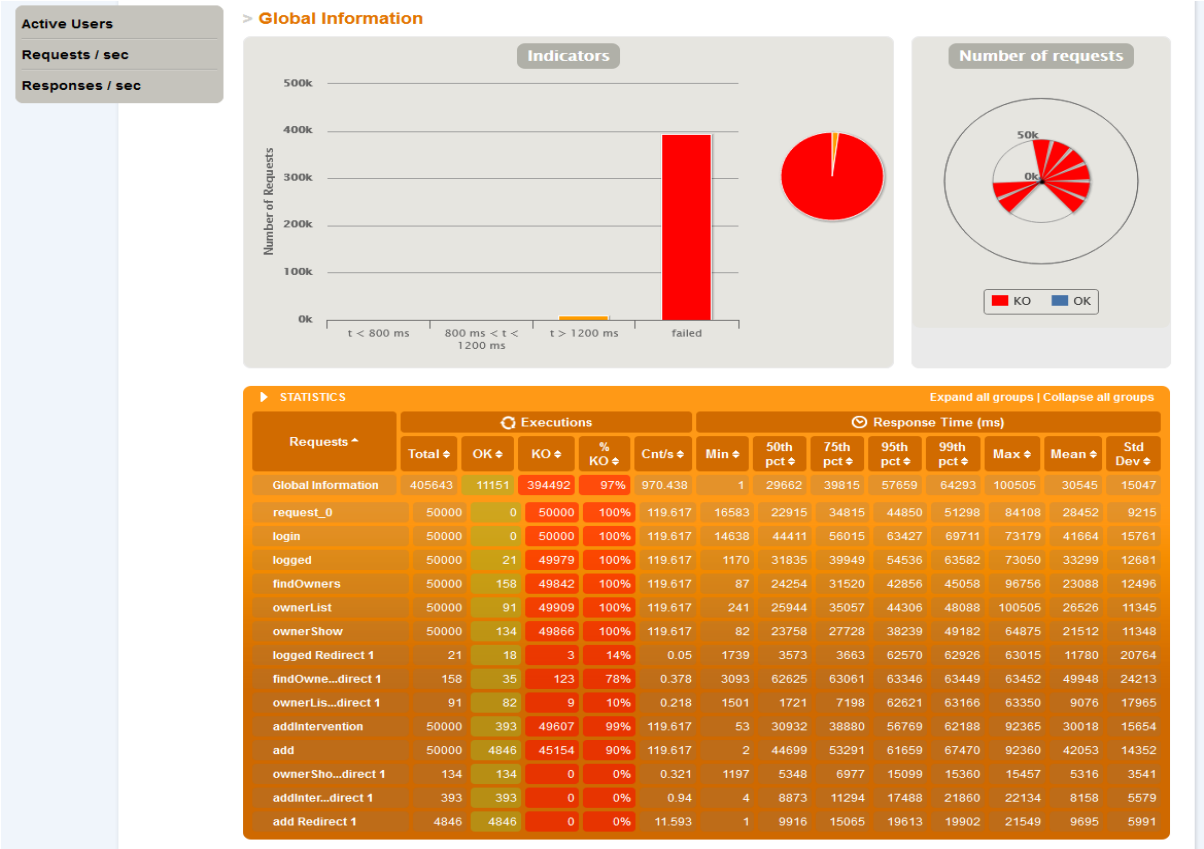


Figure 11 US-015 Stress test

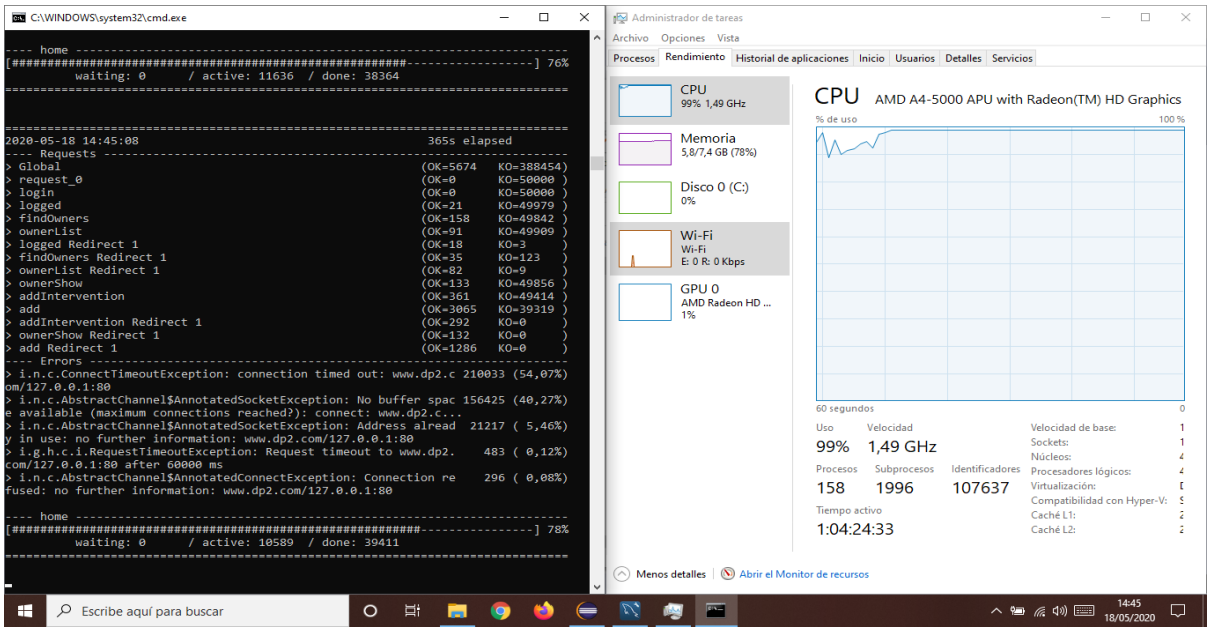
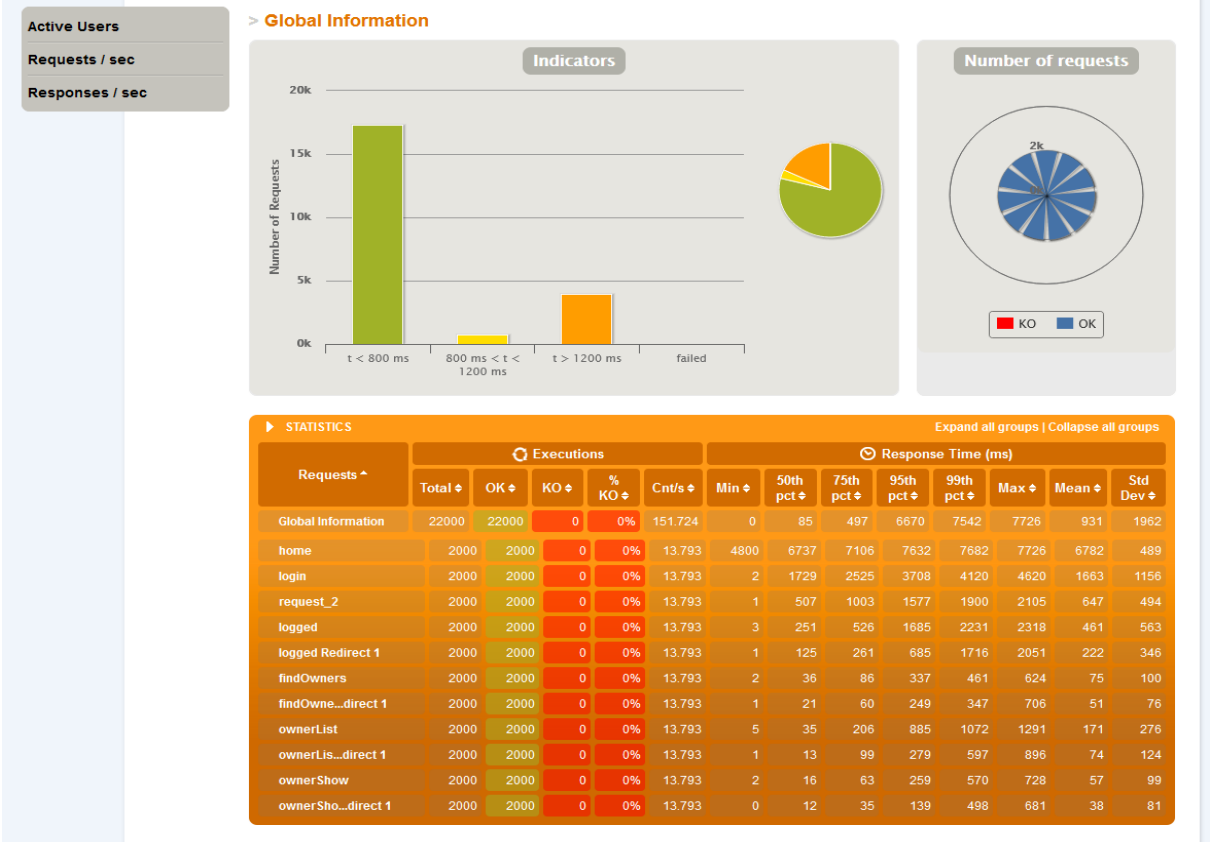


Figure 12 US-011 Stress test performance

US-16

- Load test

-2000 users. While running test with more than 2000 users, some request started failing/been rejected/taking a lot of time. (I tested adding or subtracting +-200 users per test)



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	22000	22000	0	0%	151.724	0	85	497	6670	7542	7726	931	1962
home	2000	2000	0	0%	13.793	4800	6737	7106	7632	7682	7726	6782	489
login	2000	2000	0	0%	13.793	2	1729	2525	3708	4120	4620	1663	1156
request_2	2000	2000	0	0%	13.793	1	507	1003	1577	1900	2105	647	494
logged	2000	2000	0	0%	13.793	3	251	526	1685	2231	2318	461	563
logged Redirect 1	2000	2000	0	0%	13.793	1	125	261	685	1716	2051	222	346
findOwners	2000	2000	0	0%	13.793	2	36	86	337	461	624	75	100
findOwne...direct 1	2000	2000	0	0%	13.793	1	21	60	249	347	706	51	76
ownerList	2000	2000	0	0%	13.793	5	35	206	885	1072	1291	171	276
ownerLis...direct 1	2000	2000	0	0%	13.793	1	13	99	279	597	896	74	124
ownerShow	2000	2000	0	0%	13.793	2	16	63	259	570	728	57	99
ownerSho...direct 1	2000	2000	0	0%	13.793	0	12	35	139	498	681	38	81

Figure 13 US-016 Load test

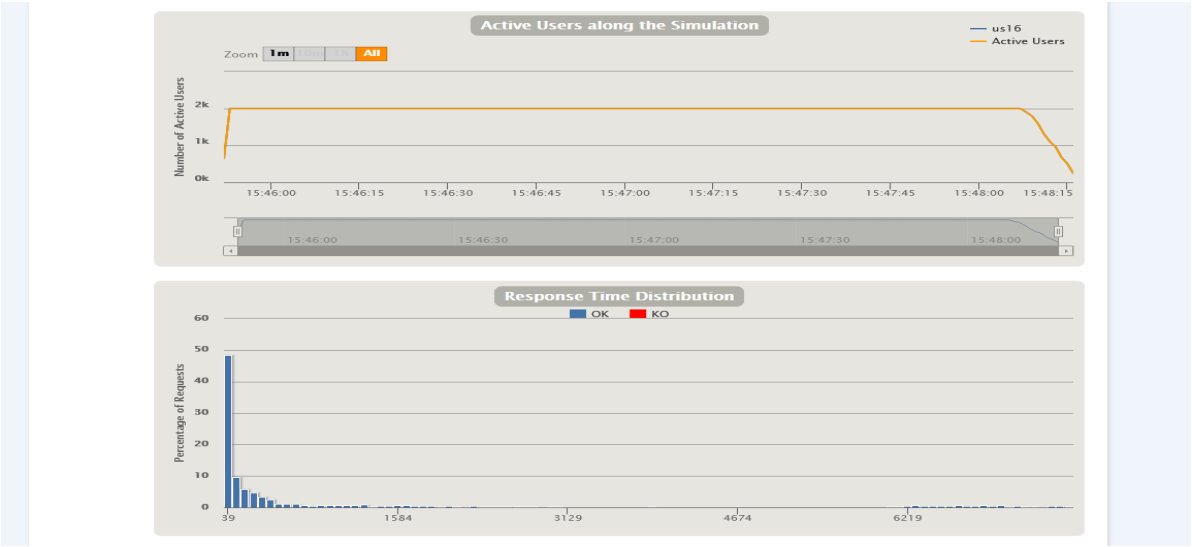
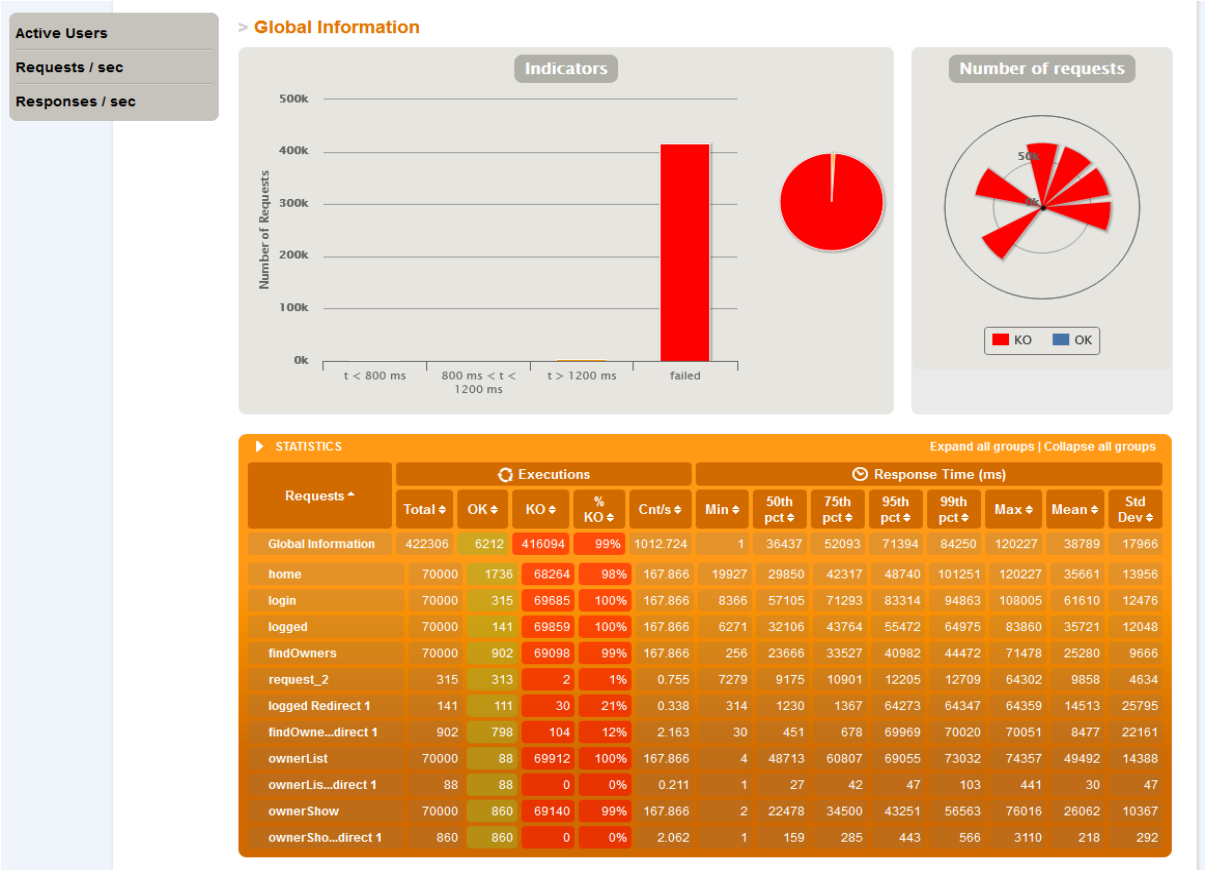


Figure 14 US-016 Load test graph

- Stress test

The failure point for this user story has been 70000 current users.



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	422306	6212	416094	99%	1012.724	1	36437	52093	71394	84250	120227	38789	17966
home	70000	1736	68264	98%	167.866	19927	29850	42317	48740	101251	120227	35661	13956
login	70000	315	69685	100%	167.866	8366	57105	71293	83314	94863	108005	61610	12476
logged	70000	141	69859	100%	167.866	6271	32106	43764	55472	64975	83860	35721	12048
findOwners	70000	902	69098	99%	167.866	256	23666	33527	40982	44472	71478	25280	9666
request_2	315	313	2	1%	0.755	7279	9175	10901	12205	12709	64302	9858	4634
logged Redirect 1	141	111	30	21%	0.338	314	1230	1367	64273	64347	64359	14513	25795
findOwne...direct 1	902	798	104	12%	2.163	30	451	678	69969	70020	70051	8477	22161
ownerList	70000	88	69912	100%	167.866	4	48713	60807	69055	73032	74357	49492	14388
ownerLis...direct 1	88	88	0	0%	0.211	1	27	42	47	103	441	30	47
ownerShow	70000	860	69140	99%	167.866	2	22478	34500	43251	56563	76016	26062	10367
ownerSho...direct 1	860	860	0	0%	2.062	1	159	285	443	566	3110	218	292

Figure 15 US-016 Stress test

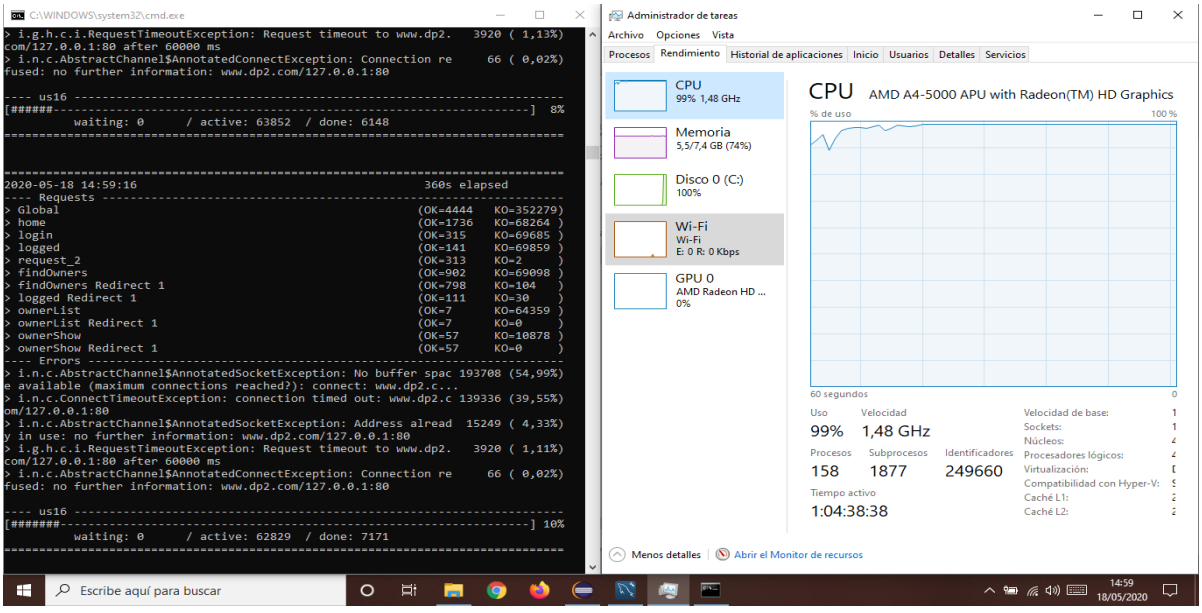


Figure 16 US-016 Stress test performance

US-17

- Load test

-2000 users. While running test with more than 2000 users, some request started failing/been rejected/taking a lot of time. (I tested adding or subtracting +-200 users per test)

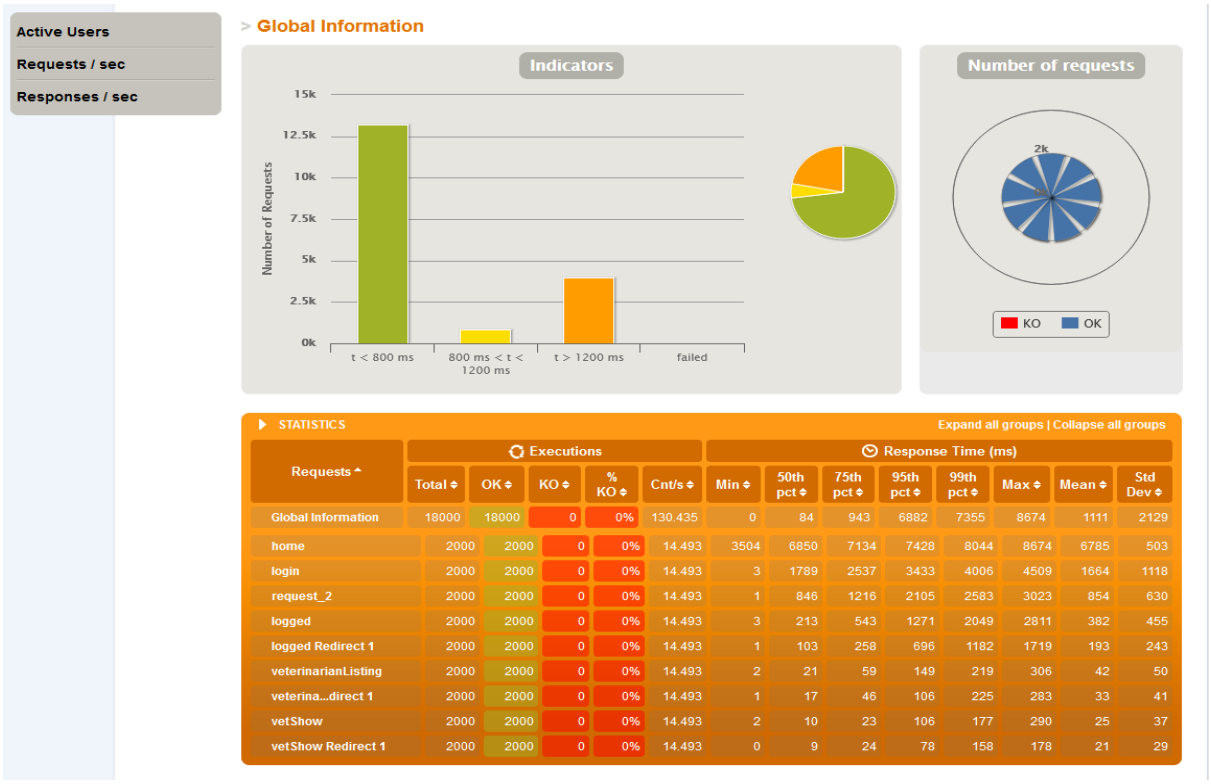


Figure 17 US-017 Load test

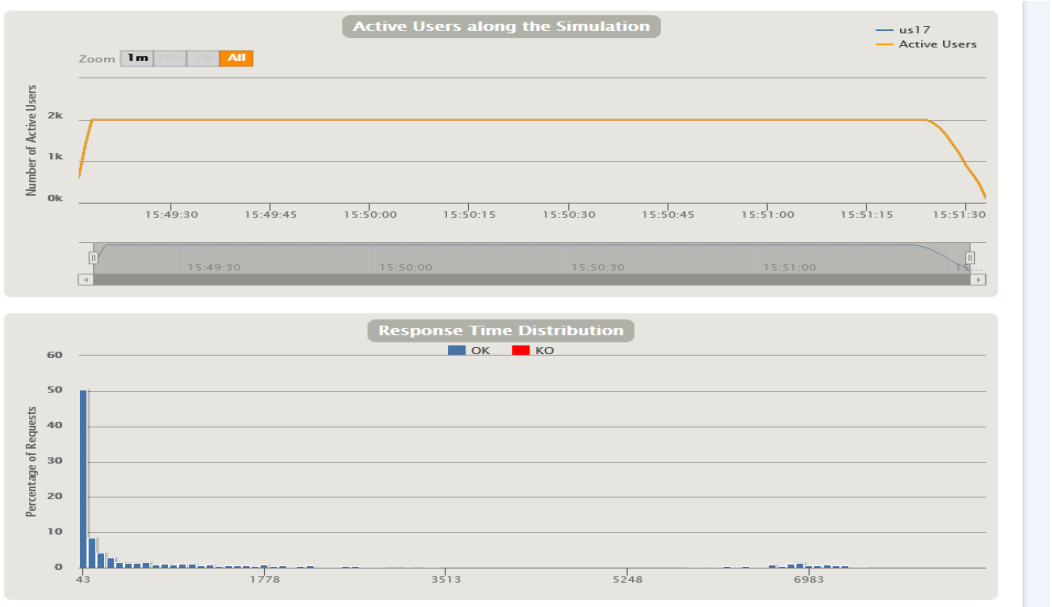


Figure 18 US-017 Load test graph

- Stress test

The failure point for this user story has been 70000 current users.

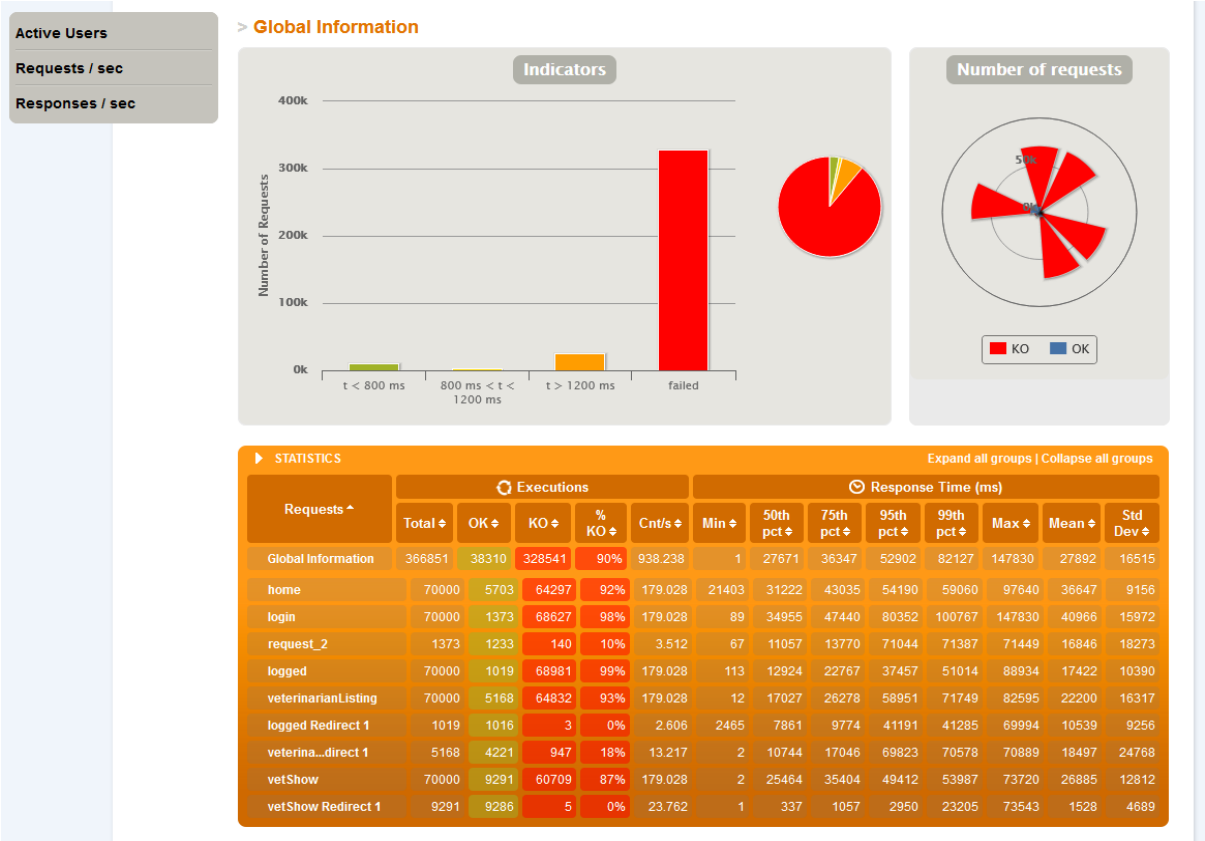


Figure 19 US-017 Stress test

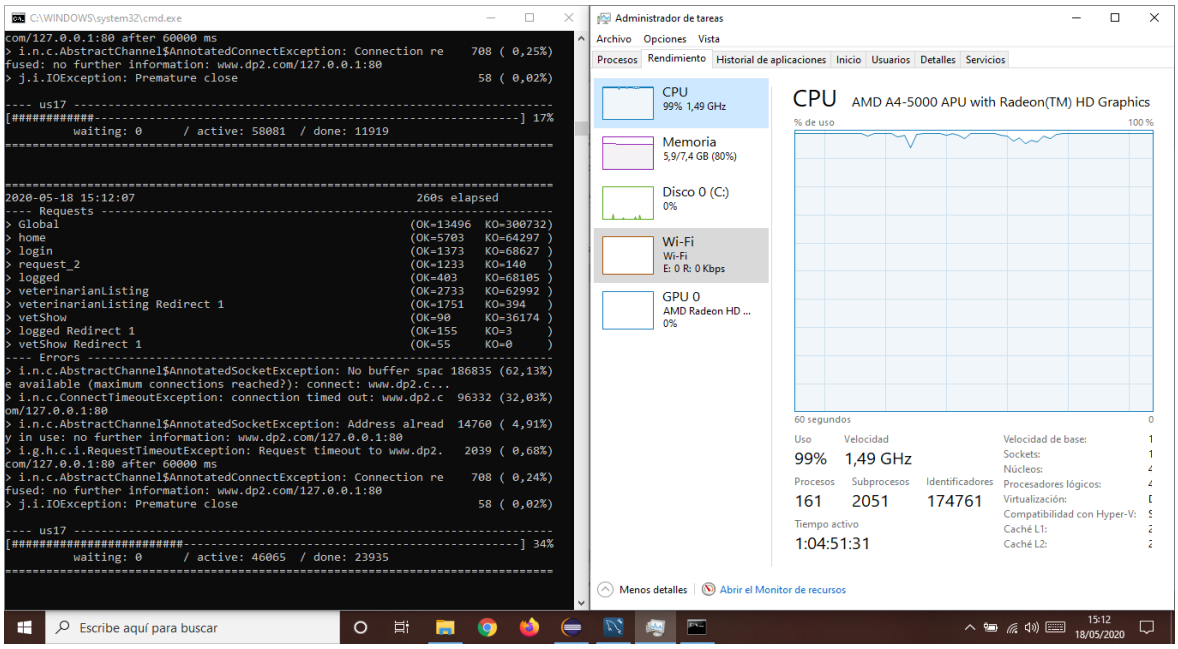


Figure 20 US-017 Stress test performance

Conclusion

US # and description	Stress test	Load test
US 010 User Adopts a Pet	70000	1500
US 014 Vet schedules new Intervention	65000	1500
US 015 Vet plans Intervention	50000	1500
US 016 Owners sees Interventions	70000	2000
US 017 Owners sees Vet’s personal information	70000	2000

As all screenshot shows, in every us test the CPU got to 99%/100& usage and the memory to 80% making the CPU the main bottle net or point failure in the system. Given that the computer used to run the test is a laptop and for that, impossible to upgrade it, the best solutions would be to purchase a new computer give significant improvement regarding CPU performance and also more memory capability.

Yoana

Throughout the whole project I have overseen the following US:

US (id)	Name
US-11	Homeless pets' management
US-13	Vet management
US-19	Trainer management
US-20	Unregistered users can see trainers
US-25	Trainers can see pet's visits

Having reached this stage of the project, performance testing was being needed to see what the basic flaws of the system are. For that we use gatling, to see how the application react to a high number of concurrent requests.

PC specs

Lenovo Legion Y520-15IKBN

Processor

- CPU Type A4
- Intel Core i7-7700HQ 2.80Ghz

Memory

- 16GB RAM

Performance Analysis

US 011 – Homeless pets' management

The system performance (CPU + RAM) when executing the stress test is detailed through the next screenshots:

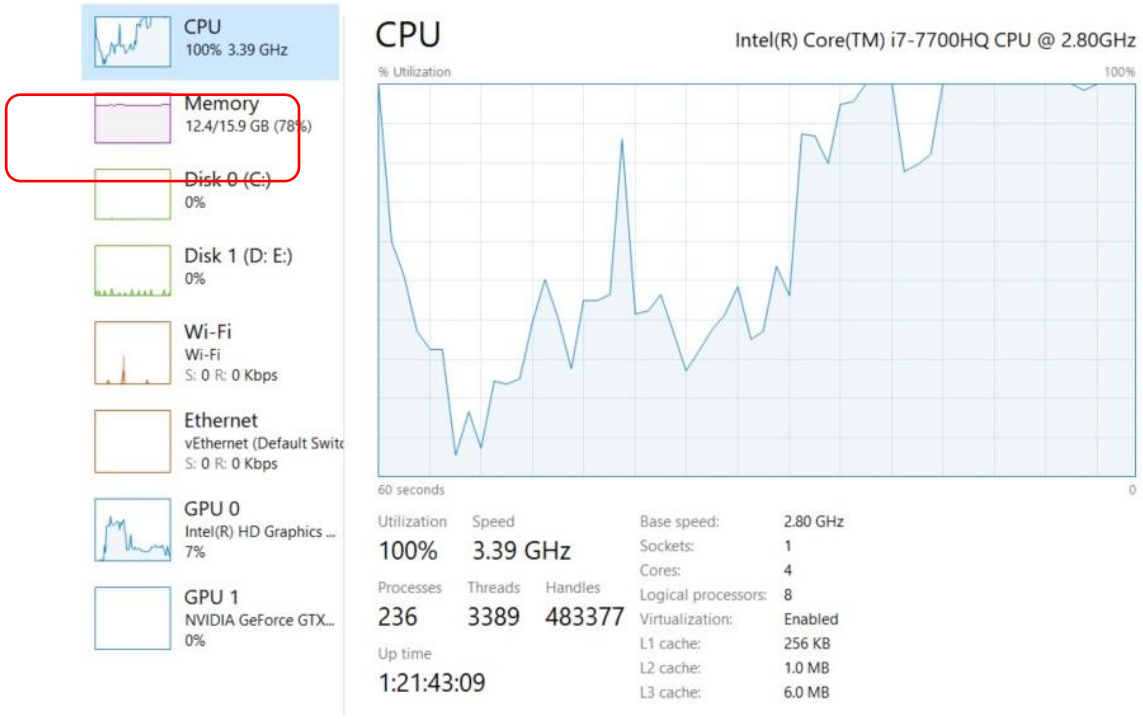


Figure 21 US 011 Bottleneck

The CPU would remain like this until the script finished, as it is shown in the next one:

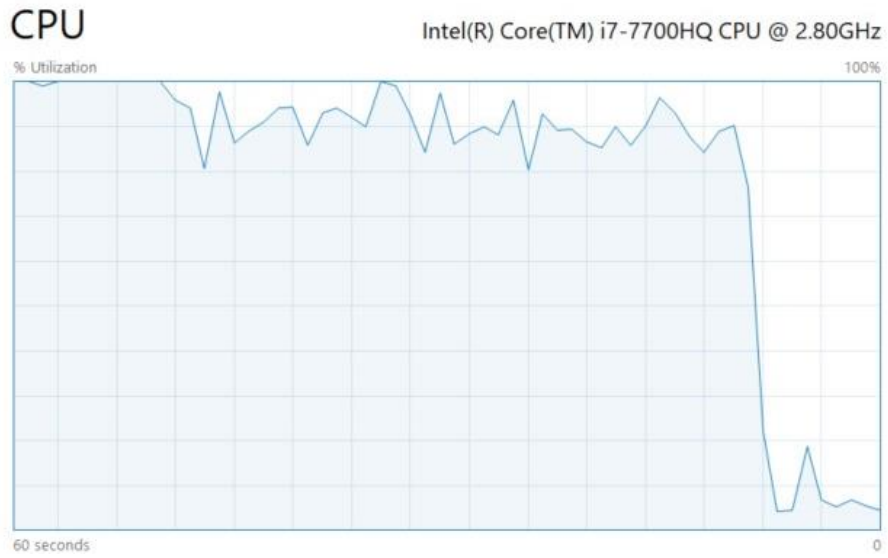


Figure 22 US 011 CPU after finishing

The Gatling report we obtain for the stress test is the following:

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	682825	5138	677687	99%	2101	0	4989	7513	15274	22451	62734	5965	4540
Home	170000	97	169903	100%	523.077	37	8376	10004	21561	23526	33365	9650	5004
Login	170000	743	169257	100%	523.077	1	5261	6766	9195	15505	26057	5717	2268
Logged	743	741	2	0%	2.286	3	4934	29309	41216	55658	58152	12839	16444
Logged Redirect 1	741	741	0	0%	2.28	1	5570	21418	28783	28789	32232	10787	10984
ListHomelessPets	170000	742	169258	100%	523.077	0	4259	5331	12885	15026	62734	4619	3945
ListHome...direct 1	67	67	0	0%	0.206	0	10909	14200	17755	18101	18638	9859	5302
Homeless...Positive	85000	428	84572	99%	261.538	0	2582	4559	12690	15227	48654	3792	3646
Homeless...Negative	85000	316	84684	100%	261.538	0	2566	4549	12507	15203	55631	3724	3519
Homeless...direct 1	30	30	0	0%	0.092	0	1	2	16172	23015	25115	1905	5888
HomelessPetCreated	429	429	0	0%	1.32	1	12621	17359	17970	45823	56433	11404	8484
Homeless...direct 1	420	410	10	2%	1.292	0	31147	31924	57743	60000	60004	23524	16347
Homeless...direct 1	39	39	0	0%	0.12	0	1	1	20	2476	3884	105	614
Homeless...rMessage	317	316	1	0%	0.975	1	9563	16843	18002	31027	51322	10232	7207
Homeless...direct 1	39	39	0	0%	0.12	1	1554	1846	27804	27805	27805	3565	7014

Figure 23 Stress test information

Regarding the load test, the Gatling report we obtain is the following:

Figure 24 US 011 Global information Load test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	4088	4009	79	2%	11.849	0	562	19732	32806	46787	60002	10026	13769
Home	500	500	0	0%	1.449	2	6	10	8376	10068	10229	1235	2720
Login	500	500	0	0%	1.449	0	2	3	8449	10015	10530	956	2406
Logged	500	500	0	0%	1.449	4	19798	30004	44588	55939	56337	18950	14930
Logged Redirect 1	500	500	0	0%	1.449	0	3	708	4002	6180	15247	846	1893
ListHomelessPets	500	497	3	1%	1.449	1	30004	30027	46579	56501	60002	24131	15790
Homeless...Positive	250	227	23	9%	0.725	1	18230	29054	30006	32524	35920	17137	11121
Homeless...Negative	250	235	15	6%	0.725	1	17596	28934	30004	31351	39129	15695	11310
ListHome...direct 1	151	151	0	0%	0.438	0	1	2	207	209	209	42	78
HomelessPetCreated	250	227	23	9%	0.725	0	3106	17088	33034	39003	47746	10848	12966
Homeless...rMessage	250	235	15	6%	0.725	0	3163	19792	32966	35513	42492	10977	12859
Homeless...direct 1	72	72	0	0%	0.209	0	2073	2408	3112	3136	3140	1393	1200
Homeless...direct 1	79	79	0	0%	0.229	0	537	2405	3113	3136	3138	1123	1180
Homeless...direct 1	207	207	0	0%	0.6	0	30003	30006	38934	45731	48687	19655	15217
Homeless...direct 1	79	79	0	0%	0.229	0	1	2	7	8	9	2	2

Figure 25 US 011 Load test details

US 013 – Vet management

The system performance (CPU + RAM) when executing the stress test is detailed through the next screenshots:

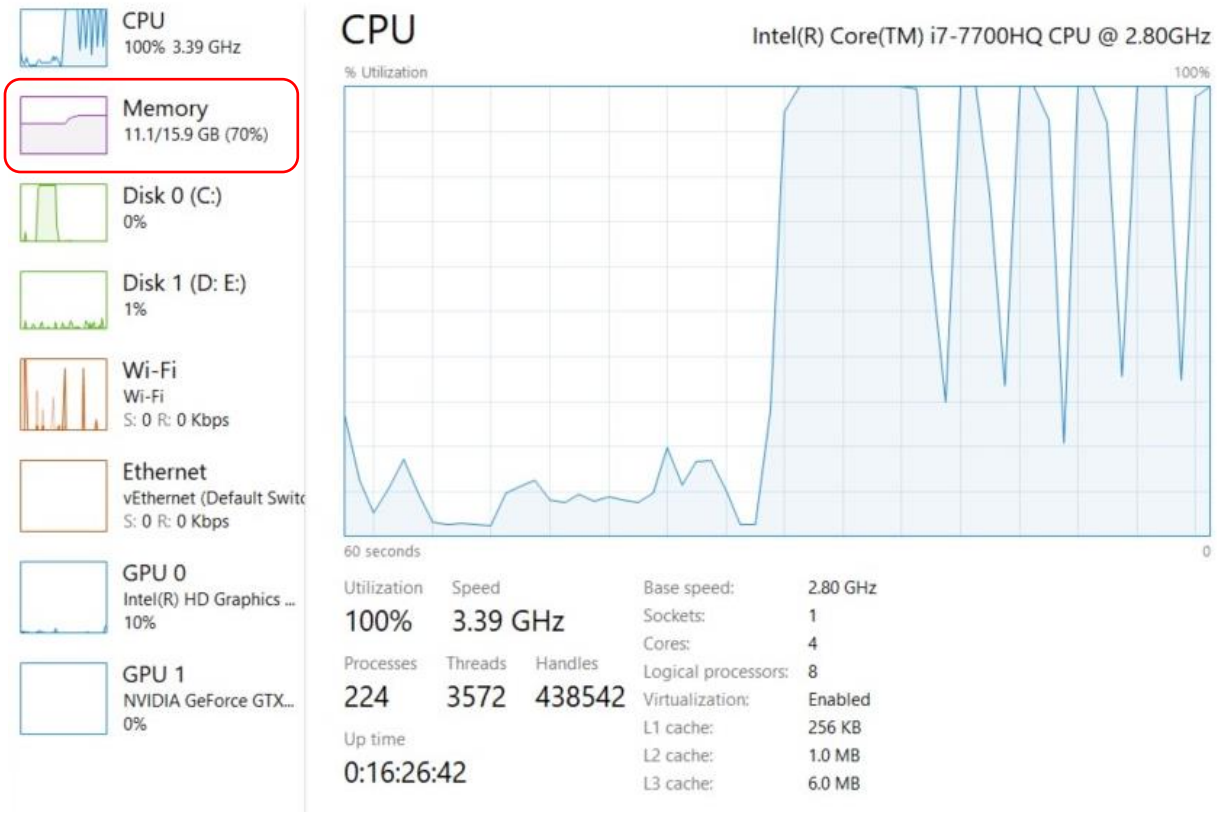


Figure 26 US 013 Bottleneck

The CPU would remain like this until the script finished, as it is shown in the next one:

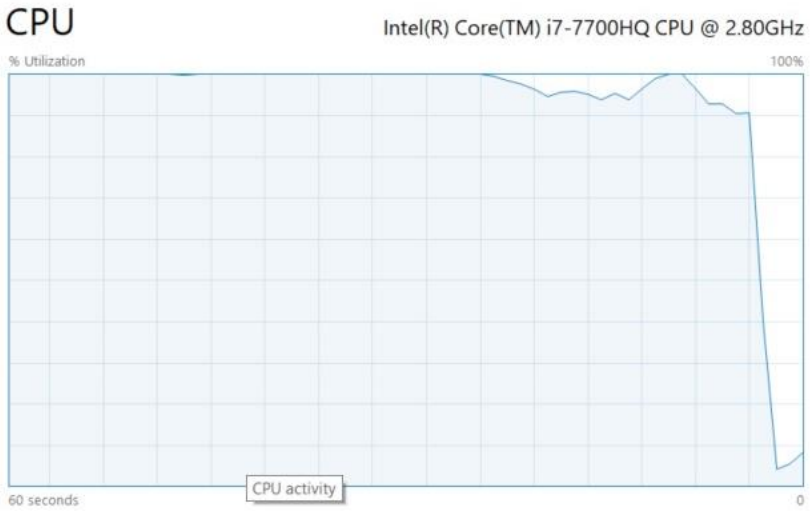


Figure 27 US 013 CPU after finishing

The Gatling report we obtain for the stress test is the following:

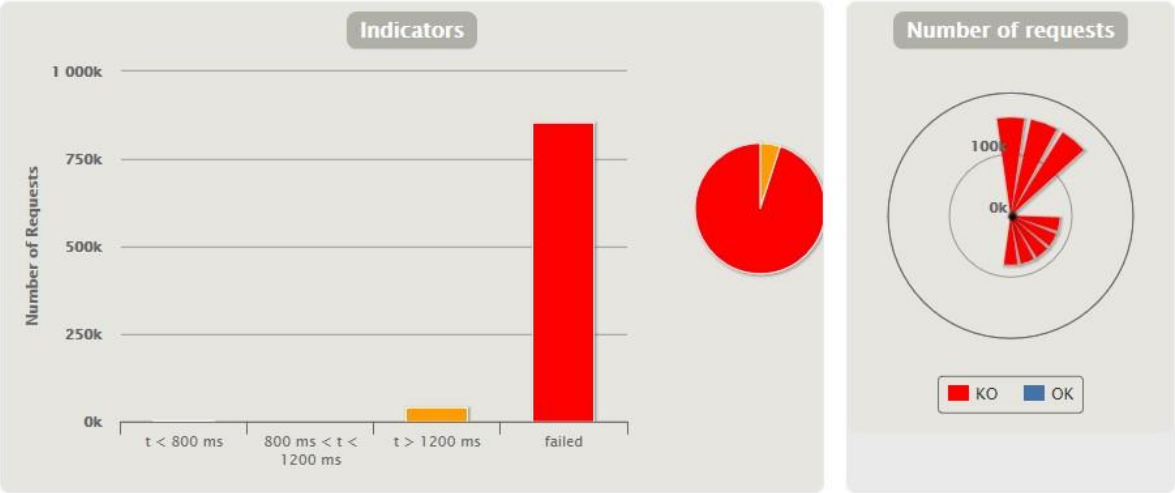


Figure 28 Global Information Stress test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	899312	44633	854679	95%	2660.686	0	3498	7071	17068	38749	65551	5326	6992
Home	160000	2905	157095	98%	473.373	2180	8633	10181	19932	23734	30453	9215	5089
Login	160000	4789	155211	97%	473.373	0	4383	8193	9158	18294	46096	5155	3482
ListVets	160000	4581	155419	97%	473.373	0	1929	3315	7173	38925	65551	3240	6671
Logged	4789	4674	115	2%	14.169	7	10734	54596	59201	61513	62525	24215	23231
Logged Redirect 1	4674	4674	0	0%	13.828	1	21594	31710	55964	56486	58524	21235	18990
ShowVetTestDummy	80000	2498	77502	97%	236.686	0	2286	4723	11778	39563	64288	4033	6932
ShowVetJamesCarter	80000	2211	77789	97%	236.686	0	2266	4632	11060	38706	64541	3930	6773
ShowVetT...ummyForm	80000	2728	77272	97%	236.686	0	2999	5044	11825	35649	65423	4030	5551
ShowVetJ...rterForm	80000	2457	77543	97%	236.686	0	3216	5058	11909	35806	62082	4082	5509
ShowVetJ...rMessage	80000	3343	76657	96%	236.686	0	1401	2861	14418	41879	60004	3758	8080
ListVets Redirect 1	125	125	0	0%	0.37	1	36973	37374	43463	44291	48884	36349	4491
ShowVetJ...direct 1	303	303	0	0%	0.896	2	23125	23190	33907	37521	47951	21958	6826
ShowVetT...direct 1	124	124	0	0%	0.367	13695	14593	32424	42956	44293	46775	22547	10223
ShowVetJ...direct 1	81	81	0	0%	0.24	13694	30794	34025	43091	44629	46775	27860	9882
ShowVetT...direct 1	337	337	0	0%	0.997	0	23081	23191	37355	44093	44354	20559	9347
VetTestDummyUpdated	2806	2730	76	3%	8.302	0	14007	16651	31346	37353	60001	14898	8355
ShowVetJ...direct 1	3343	3343	0	0%	9.891	0	23075	23145	34099	37020	37616	22339	5771
VetTestD...direct 1	2730	2730	0	0%	8.077	0	10821	14734	16803	18773	35570	11045	4538

Figure 29 Stress test information details

Regarding the load test, the Gatling report we obtain is the following:

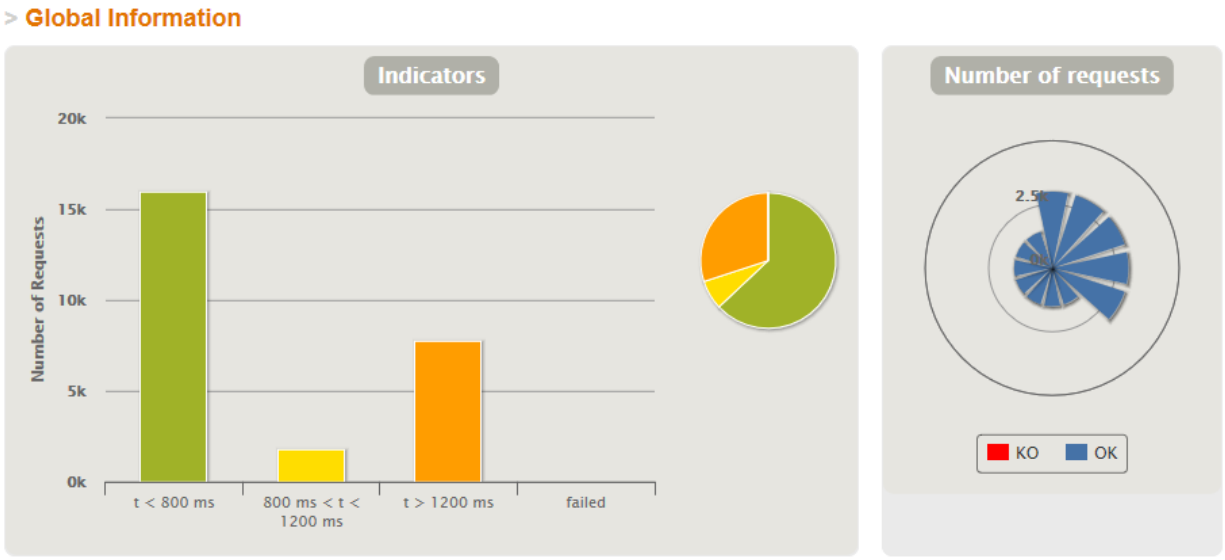


Figure 30 US 013 Global information Load test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	25500	25500	0	0%	122.596	0	116	1619	3128	4265	8150	889	1196
Home	3000	3000	0	0%	14.423	2	4	6	1107	1232	1291	145	349
Login	3000	3000	0	0%	14.423	0	2	3	1236	1615	1792	214	447
Logged	3000	3000	0	0%	14.423	2	204	2079	3499	4655	8150	980	1279
Logged Redirect 1	3000	3000	0	0%	14.423	1	3	1089	1579	1750	1800	408	601
ListVets	3000	3000	0	0%	14.423	55	701	2609	3558	4718	6608	1347	1345
ShowVetJamesCarter	1500	1500	0	0%	7.212	22	748	2432	3489	4730	7427	1279	1310
ShowVetTestDummy	1500	1500	0	0%	7.212	11	759	2391	3285	4586	6380	1230	1276
ShowVetT...ummyForm	1500	1500	0	0%	7.212	12	768	2393	3321	4212	6433	1241	1252
ShowVetJ...rterForm	1500	1500	0	0%	7.212	22	867	2457	3464	4629	7116	1316	1298
VetTestDummyUpdated	1500	1500	0	0%	7.212	13	646	2431	3538	4591	6052	1261	1345
VetTestD...direct 1	1500	1500	0	0%	7.212	50	715	2555	3542	4724	7600	1339	1365
ShowVetJ...rMessage	1500	1500	0	0%	7.212	7	710	2417	3505	4778	6237	1256	1350

Figure 31 US 013 Load test details

US 019 – Trainer management

The system performance (CPU + RAM) when executing the stress test is detailed through the next screenshots:

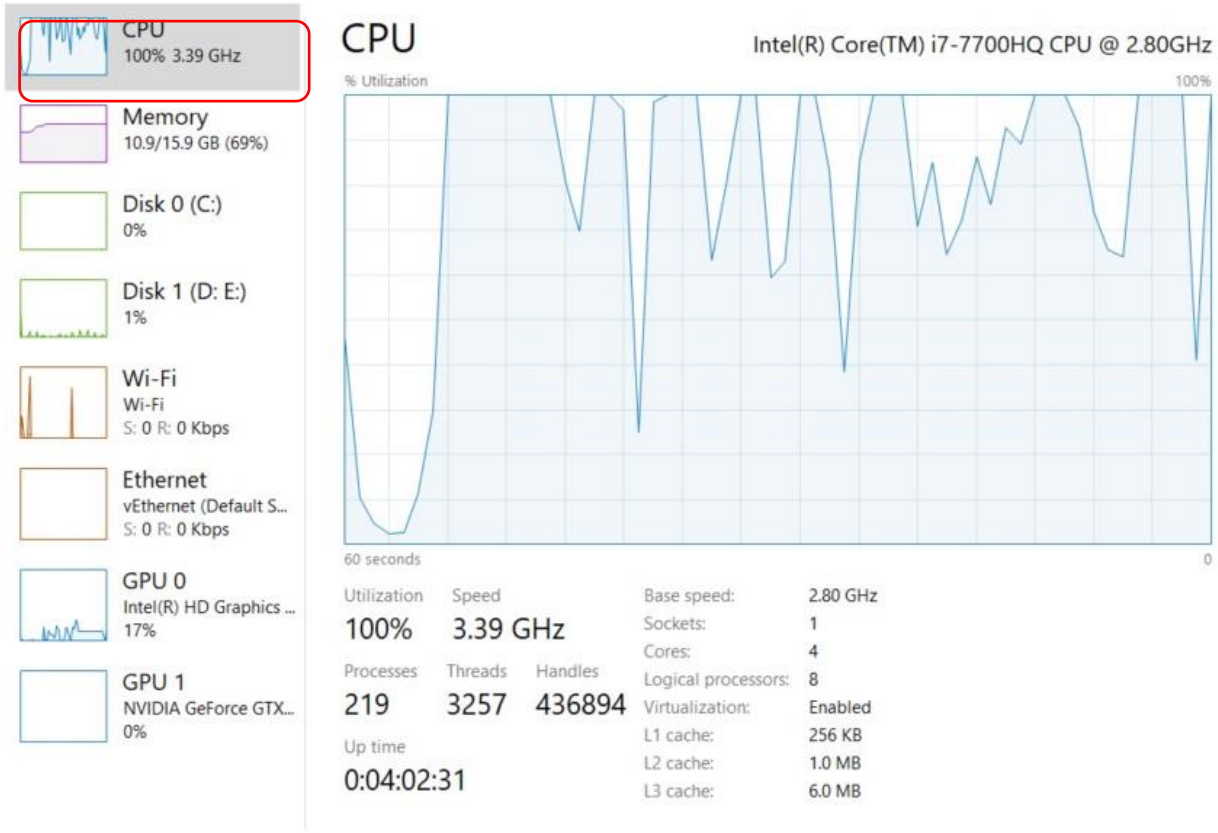


Figure 32 US 019 Bottleneck

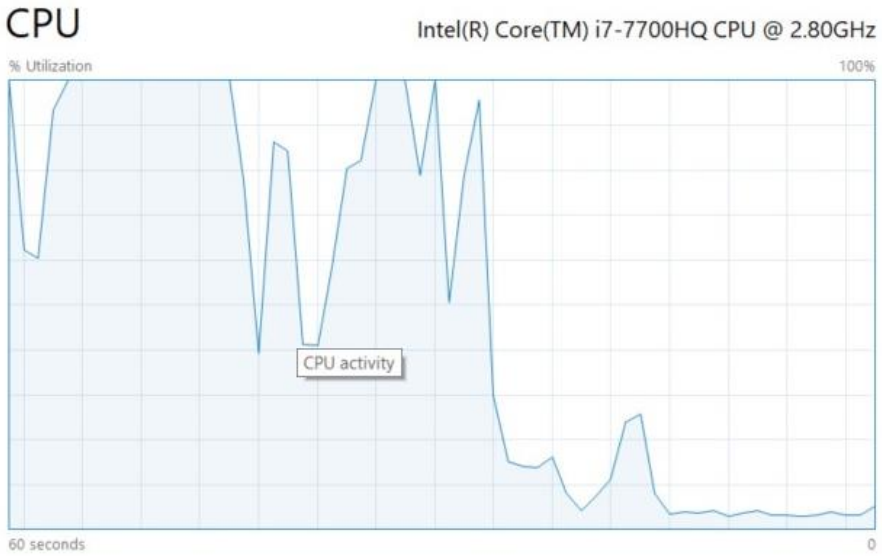
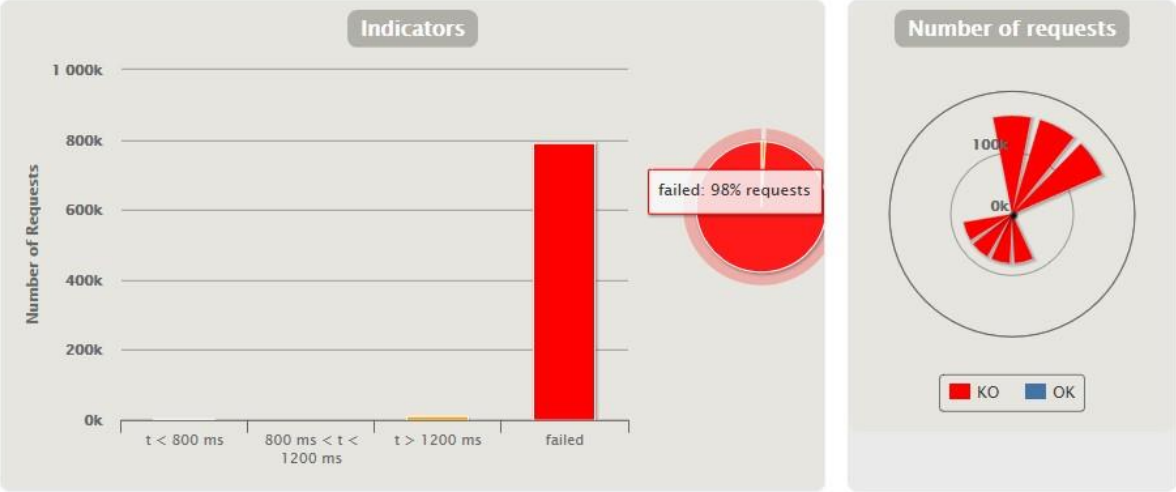


Figure 33 US 019 after finishing

The Gatling report we obtain for the stress test is the following:



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	804923	15066	789857	98%	4547.588	0	6295	9238	17404	20743	44256	6800	4685
Home	160000	472	159528	100%	903.955	1721	6496	8549	19586	21730	31510	7978	4855
Login	160000	2374	157626	99%	903.955	0	8043	9654	11821	20997	22227	8306	3462
ListTrainers	160000	2375	157625	99%	903.955	0	4525	7107	15355	19743	44256	5234	4544
Logged	2374	2374	0	0%	13.412	62	6087	9084	30453	33554	44054	9899	9242
Logged Redirect 1	2374	2374	0	0%	13.412	53	1211	2079	9433	27515	32981	2452	3805
ListTrai...direct 1	1	1	0	0%	0.006	24	24	24	24	24	24	24	0
ShowTrai...omasMoon	80000	1191	78809	99%	451.977	0	5704	9662	16881	19670	24516	6508	5030
ShowTrainerJohnDoe	80000	1184	78816	99%	451.977	0	5709	9660	16875	19613	24516	6510	5036
TrainerNotDeleted	80000	1210	78790	98%	451.977	0	5617	9390	13459	19513	24139	6103	4525
TrainerD...essfully	80000	1337	78663	98%	451.977	0	5140	9521	13131	19293	24098	5891	4439
ShowTrai...direct 1	1	1	0	0%	0.006	400	400	400	400	400	400	400	0
TrainerD...direct 1	147	147	0	0%	0.831	0	3	4	7	36	736	8	60
TrainerN...direct 1	26	26	0	0%	0.147	0	2	2	3	3	3	2	1

Figure 34 Stress test information

Regarding the load test, the Gatling report we obtain is the following:

Figure 35 US 019 Global information Load test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	42000	42000	0	0%	217.617	0	821	2053	3544	4414	8418	1152	1270
Home	6000	6000	0	0%	31.088	1	6	1130	2348	2824	2931	584	808
Login	6000	6000	0	0%	31.088	0	5	1150	2180	2690	2937	596	769
Logged	6000	6000	0	0%	31.088	2	1145	2405	3776	4952	7763	1366	1396
Logged Redirect 1	6000	6000	0	0%	31.088	1	580	1610	2637	2909	3036	871	937
ListTrainers	6000	6000	0	0%	31.088	16	1386	2496	3710	4641	8418	1480	1349
ShowTrainerJohnDoe	3000	3000	0	0%	15.544	14	1678	2926	3974	4897	8265	1668	1456
ShowTrai...omasMoon	3000	3000	0	0%	15.544	5	1658	2866	3849	4873	7745	1635	1440
TrainerNotDeleted	3000	3000	0	0%	15.544	19	1348	2578	3794	4699	7489	1486	1398
TrainerD...essfully	3000	3000	0	0%	15.544	19	1388	2770	3998	5132	7084	1551	1473

Figure 36 US 019 Load test details

US 020 – Unregistered users can see trainers

The system performance (CPU + RAM) when executing the stress test is detailed through the next screenshots:

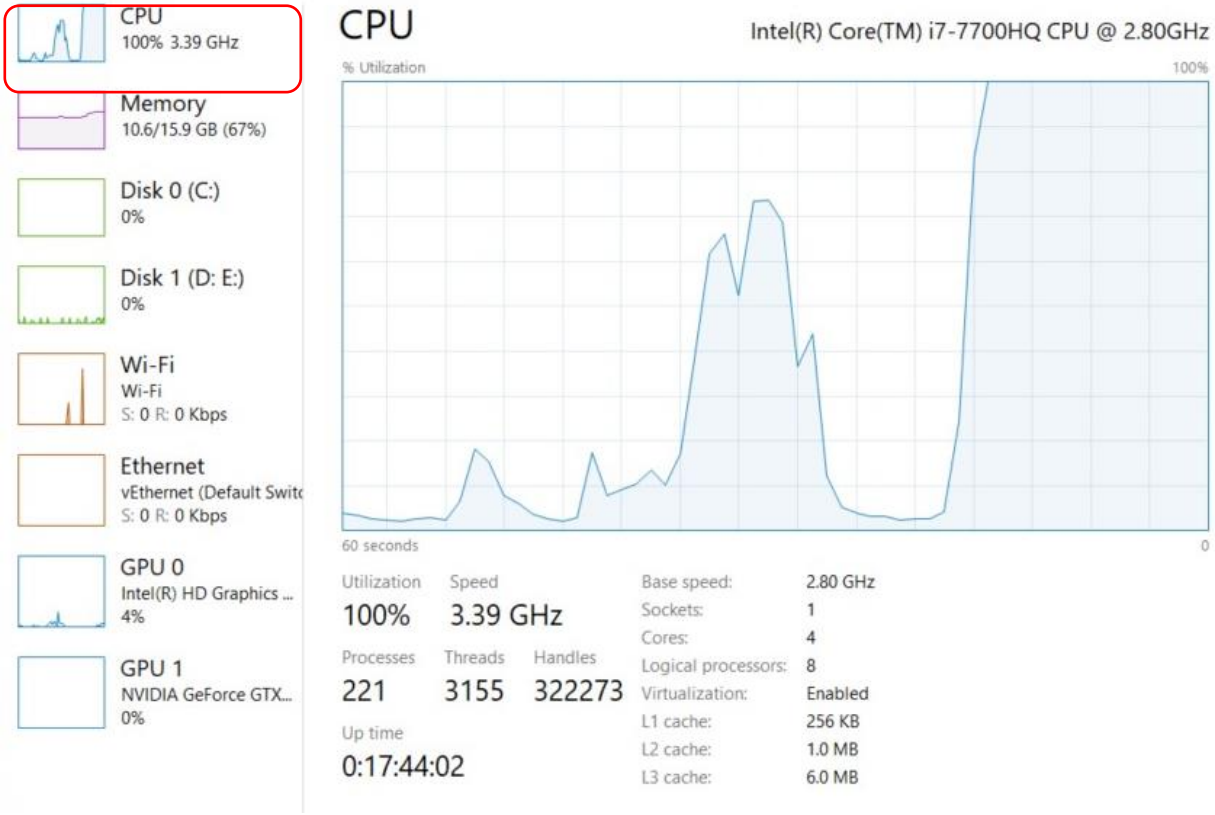


Figure 37 US 020 Bottleneck

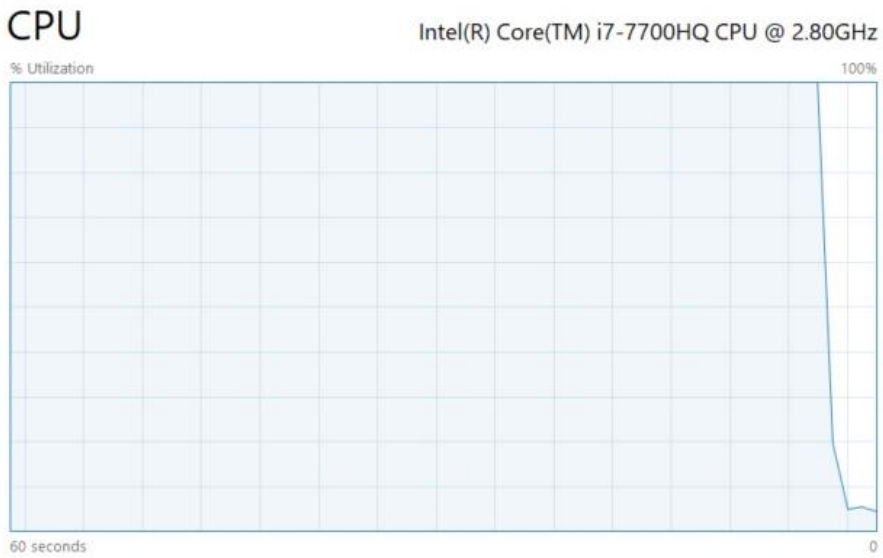
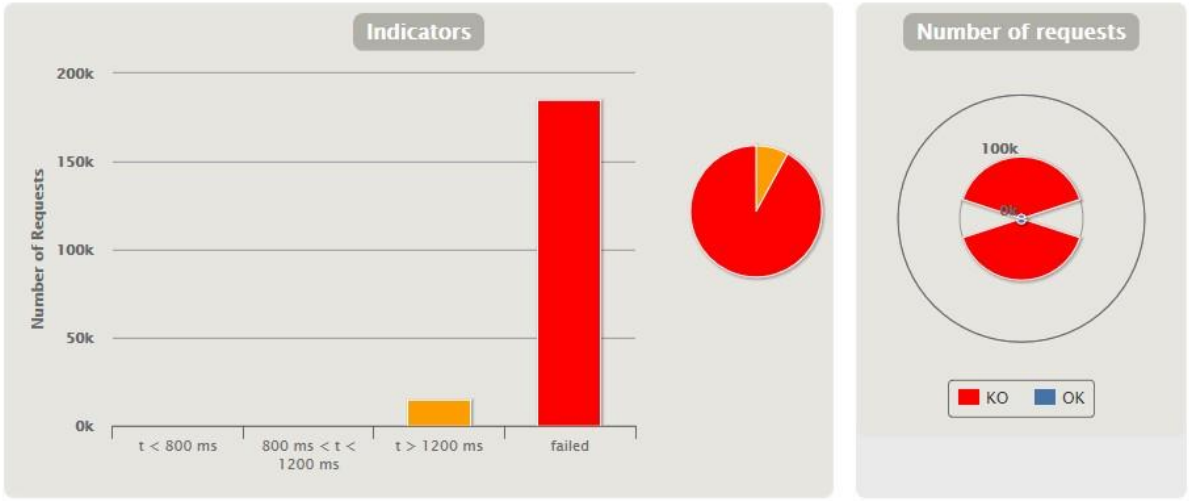


Figure 38 US 020 CPU after finishing

The Gatling report we obtain for the stress test is the following:



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	200000	15147	184853	92%	2380.952	0	1505	2354	11969	43824	60490	3501	7353
Home	100000	6891	93109	93%	1190.476	2	1701	2374	11209	13413	56578	3105	4063
ListTrainers	100000	8256	91744	92%	1190.476	0	1032	2326	29329	48314	60490	3897	9556

Figure 39 Stress test information

Regarding the load test, the Gatling report we obtain is the following:

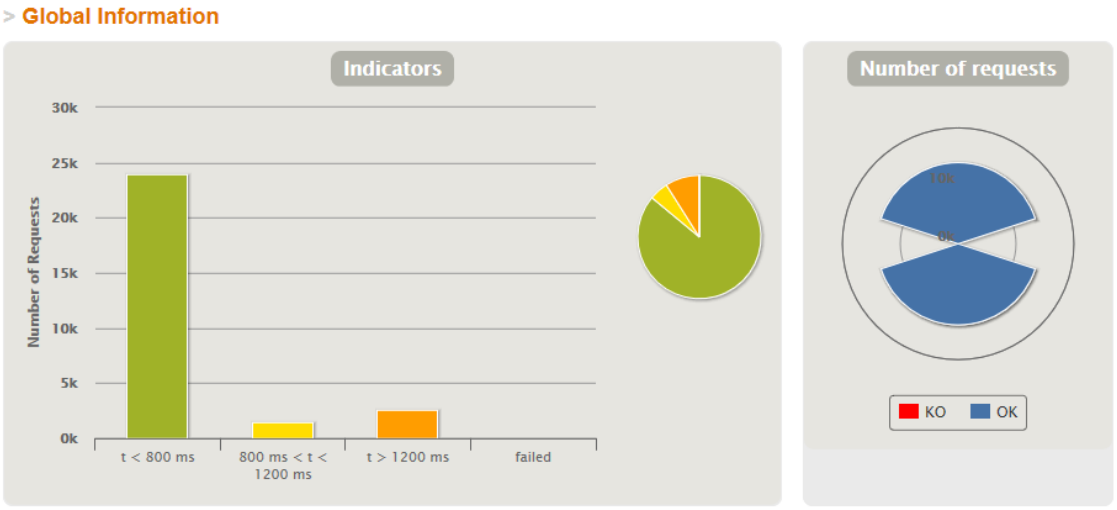


Figure 40 US 020 Global information Load test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	28000	28000	0	0%	250	1	31	348	1880	2618	5337	332	637
Home	14000	14000	0	0%	125	1	4	5	947	1209	1305	110	291
ListTrainers	14000	14000	0	0%	125	19	132	678	2484	2878	5337	553	794

Figure 41 US 020 Load test details

US 025 – Trainers can see pets' visits

The system performance (CPU + RAM) when executing the stress test is detailed through the next screenshots:

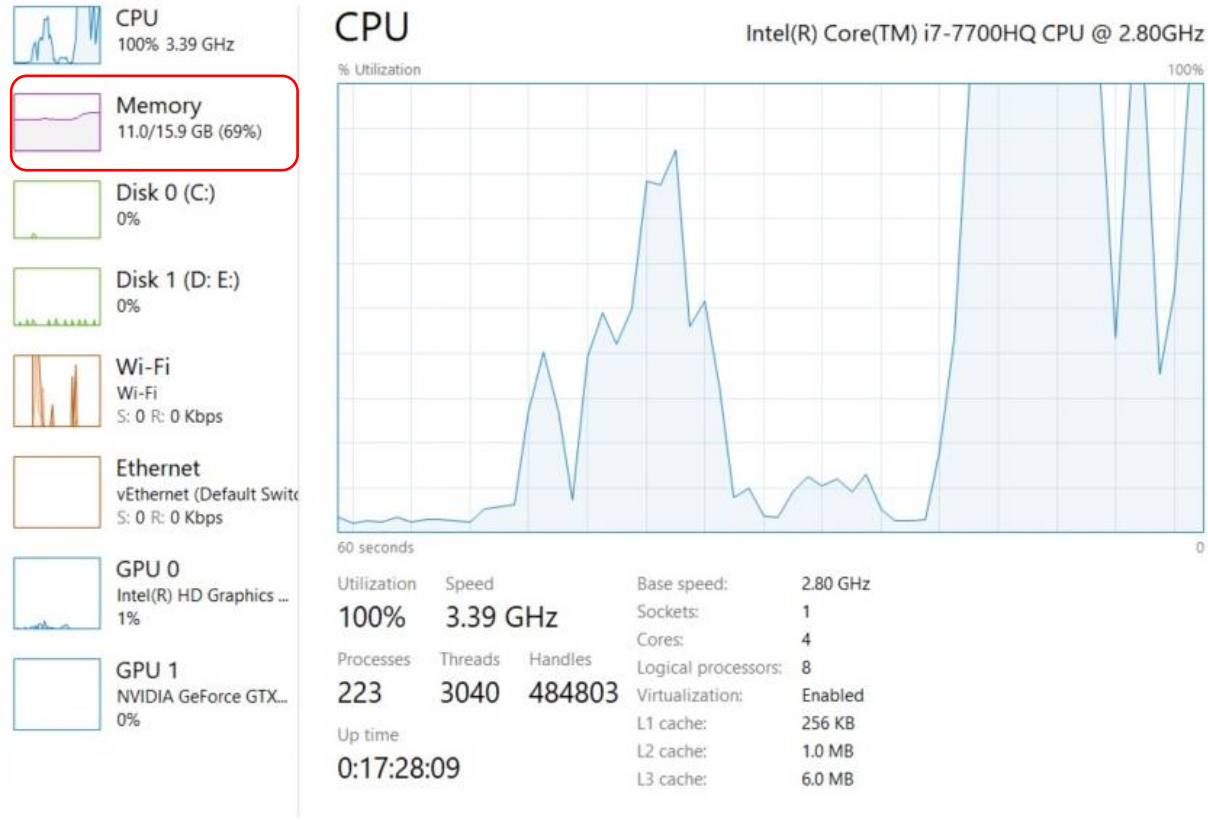


Figure 42 US 025 Bottleneck

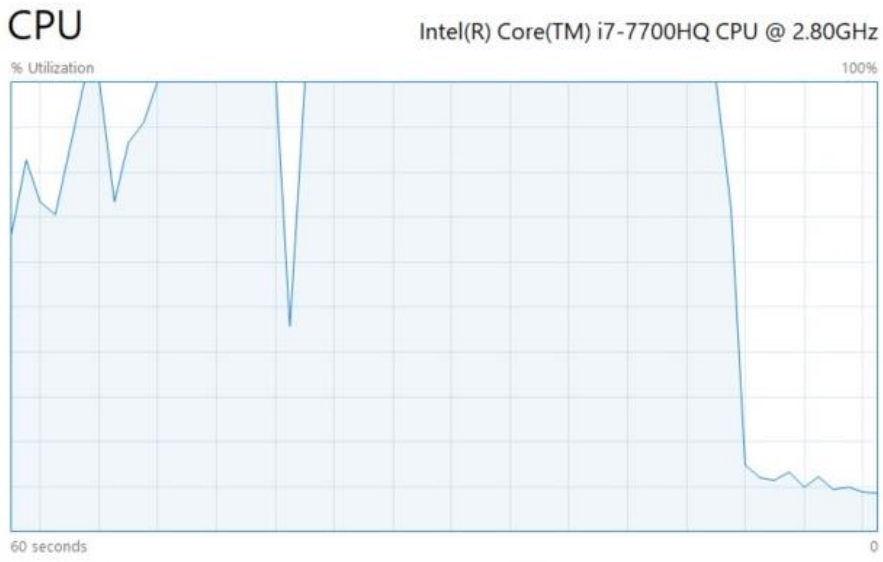
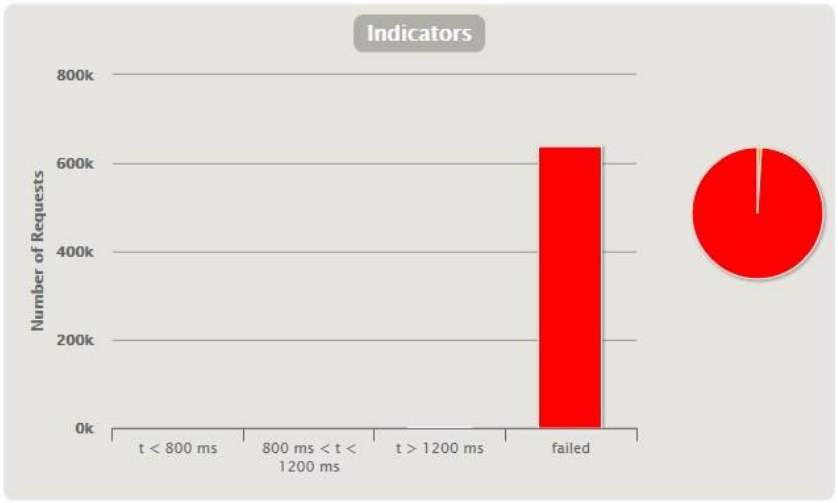
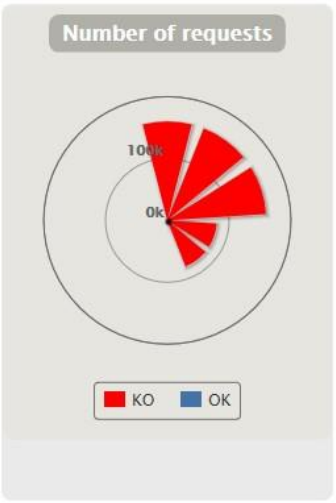


Figure 43 US 025 CPU after finishing

The Gatling report we obtain for the stress test is the following:





STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	641583	4058	637525	99%	5091.929	0	3953	7473	12812	19453	34671	4866	4164
Home	160000	110	159890	100%	1269.841	62	6065	8100	18662	20238	34671	7369	4770
Login	160000	782	159218	100%	1269.841	0	7679	9342	10792	19492	30574	7625	3343
ListHomelessPets	160000	788	159212	100%	1269.841	0	1548	2911	5249	7961	23098	1975	1784
ShowNoHo...etVisits	80000	377	79623	100%	634.921	0	2063	3733	6792	8149	20519	2525	2134
ShowHome...etVisits	80000	418	79582	99%	634.921	0	2050	3726	6710	8151	24416	2504	2122
Logged	782	782	0	0%	6.206	11	2739	4818	9767	18469	26855	3627	3450
Logged Redirect 1	782	782	0	0%	6.206	2	1811	2863	7807	11116	13777	2458	2372
ListHome...direct 1	6	6	0	0%	0.048	147	3871	6878	7580	7699	7729	3816	3248
ShowNoHo...direct 1	8	8	0	0%	0.063	1	339	780	5536	7290	7729	1338	2453
ShowHome...direct 1	5	5	0	0%	0.04	1	338	5964	7375	7657	7728	2874	3293

Figure 44 Stress test information

Regarding the load test, the Gatling report we obtain is the following:

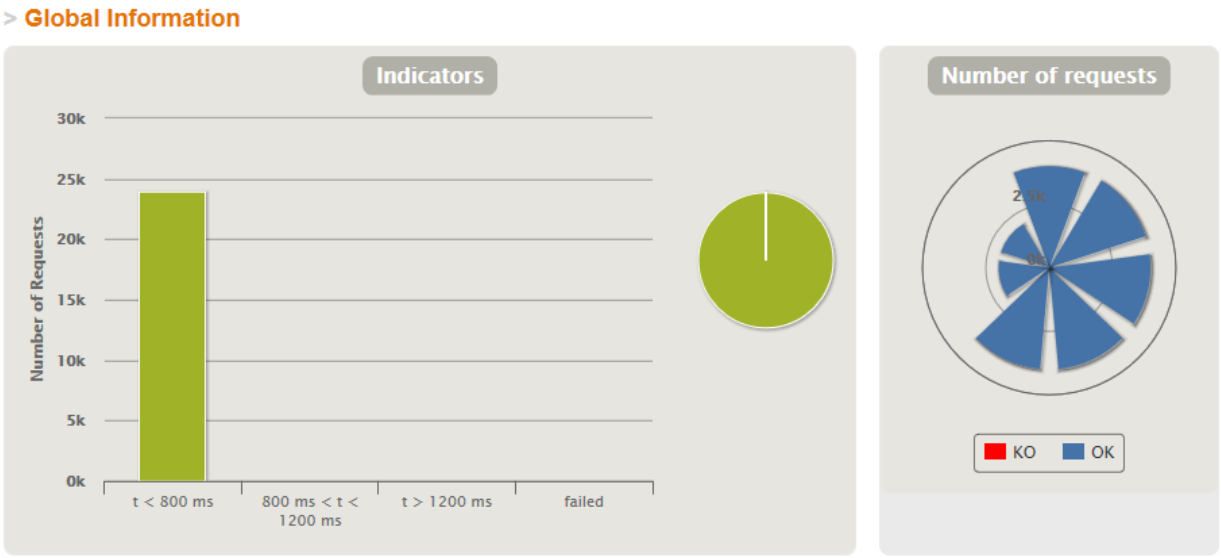


Figure 45 US 025 Global information Load test

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	24000	24000	0	0%	150.943	0	4	31	126	459	1837	31	88
Home	4000	4000	0	0%	25.157	2	3	4	6	9	62	4	2
Login	4000	4000	0	0%	25.157	0	1	2	3	6	125	2	3
Logged	4000	4000	0	0%	25.157	2	5	7	156	610	1542	31	110
Logged Redirect 1	4000	4000	0	0%	25.157	1	3	3	4	6	17	3	1
ListHomelessPets	4000	4000	0	0%	25.157	29	42	61	285	707	1837	84	126
ShowHome...etVisits	2000	2000	0	0%	12.579	21	32	44	234	693	1183	67	115
ShowNoHo...etVisits	2000	2000	0	0%	12.579	13	21	30	204	636	1269	52	109

Figure 46 US 025 Load test details

Conclusion

US # and description	Stress test	Load test
US 011 Homeless pets management	85000	250
US 013 Vet management	80000	1500
US 019 Trainer management	80000	3000
US 020 Unregistered users can see trainers	100000	14000
US 025 Trainers can see pet’s visits	80000	2000

As a conclusion of all these tests, I can say that we would have to improve mostly the CPU and in some cases the memory, but it is not that big of an issue like the CPU.

Diāna

I have been assigned to 5 user stories; therefore, performance testing has been done to each of those user stories:

US (id)	Name
US-18	Vet sees pet’s visits
US-21	Training session organization
US-22	Trainer plans rehabilitation sessions
US-23	Owner sees rehabilitation sessions
US-24	Owner sees trainer’s personal information

I have considered stress test at the lowest point in which my pc could not handle more request and gatling give me 100% error, and load test at the maximum point in which every assertion is passed.

PC specs

VivoBook 15_ASUS Laptop X540BA.

Processor

- AMD A9-9425 RADEON R5

Memory

- Max Supported Size 8 GB

Performance Analysis

US – 018 Vet sees pet's visits

Table shows results after 2 tests:

Stress testing	60000
Load testing	2000

Gatling report after performing load testing:

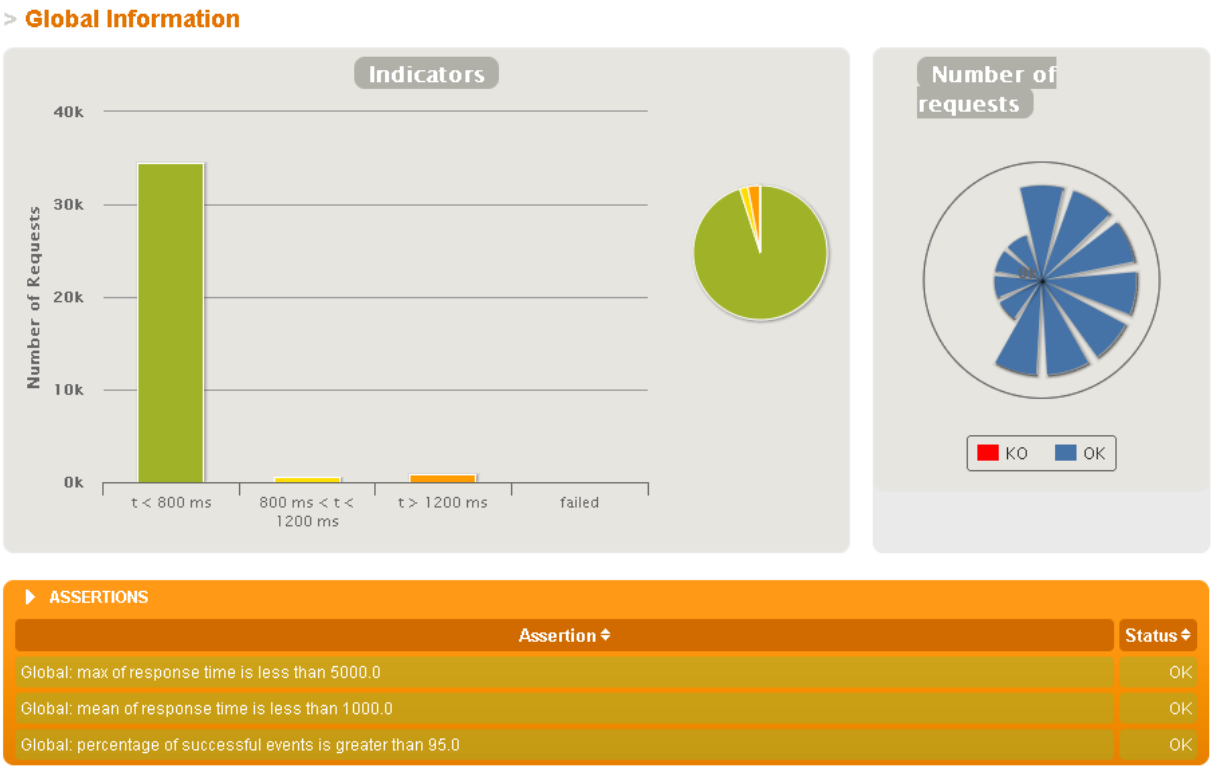


Figure 47 US-18 testing overview

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	36000	36000	0	0%	197.802	0	6	81	710	1840	4921	138	379
Home	4000	4000	0	0%	21.978	3	12	140	671	1719	1897	149	309
Login	4000	4000	0	0%	21.978	1	7	199	724	1043	4921	148	274
request_2	4000	4000	0	0%	21.978	0	3	9	434	3510	3522	118	482
Logged	4000	4000	0	0%	21.978	2	9	252	1144	2062	3972	214	427
Logged Redirect 1	4000	4000	0	0%	21.978	0	4	32	425	1531	4414	98	355
FindOwners	4000	4000	0	0%	21.978	1	7	150	1079	2095	4269	184	422
FindOwne...direct 1	4000	4000	0	0%	21.978	0	3	9	305	1135	3908	70	333
BrokenOwnersLink	2000	2000	0	0%	10.989	1	7	146	1070	2128	3826	187	427
BrokenOw...direct 1	2000	2000	0	0%	10.989	0	3	9	321	1180	3494	72	314
OwnersIn...tsVisits	2000	2000	0	0%	10.989	1	7	154	981	1973	3487	179	392
OwnersIn...direct 1	2000	2000	0	0%	10.989	0	3	9	334	1285	3837	78	335

Figure 48 US-18 testing steps

System performance during stress testing:

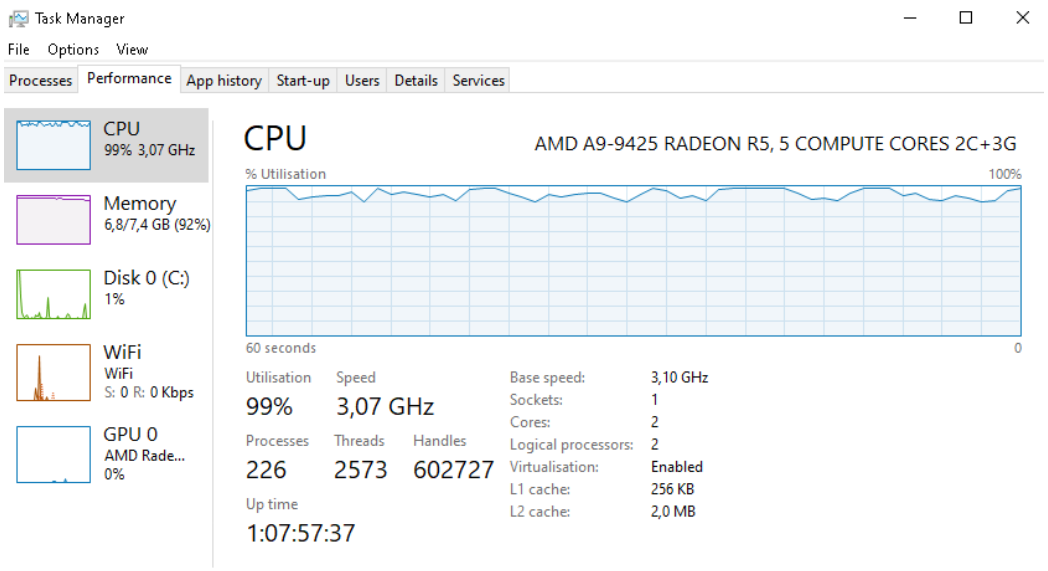


Figure 49 U-18 system performance during stress testing

US-021 Training session organization

Table shows results after 2 tests:

Stress testing	45000
Load testing	1200

Gatling report after performing load testing:

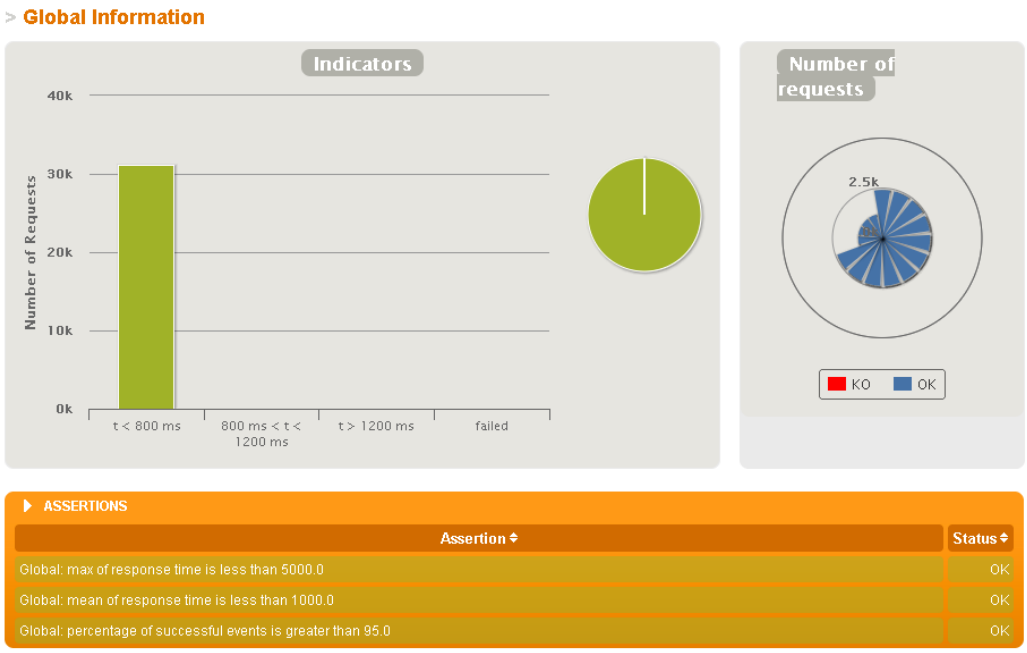


Figure 50 US-21 Testing overview

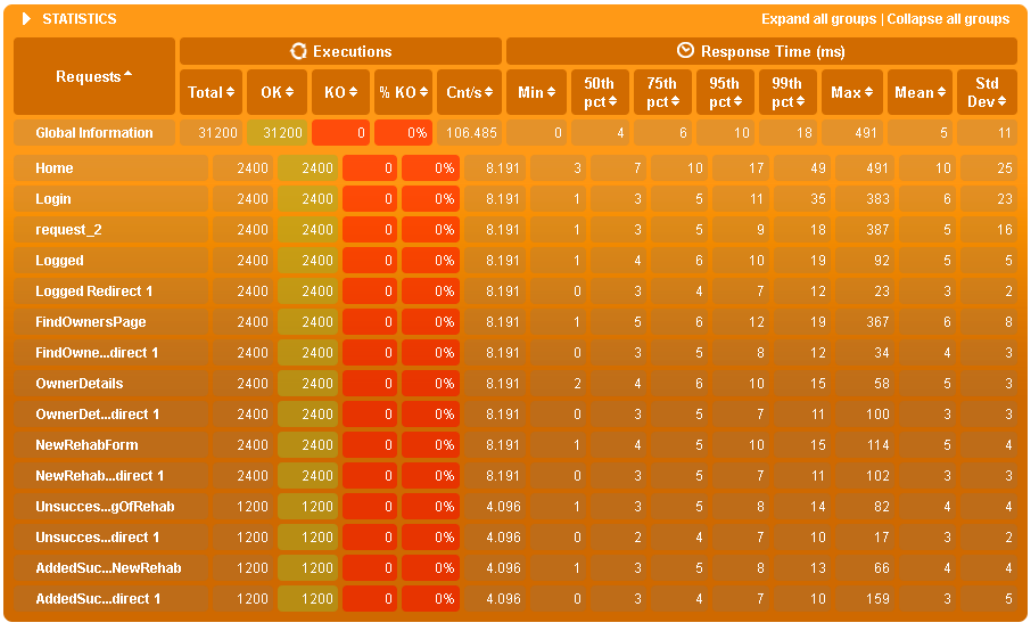


Figure 51 US-21 Testing overview

System performance during stress testing:

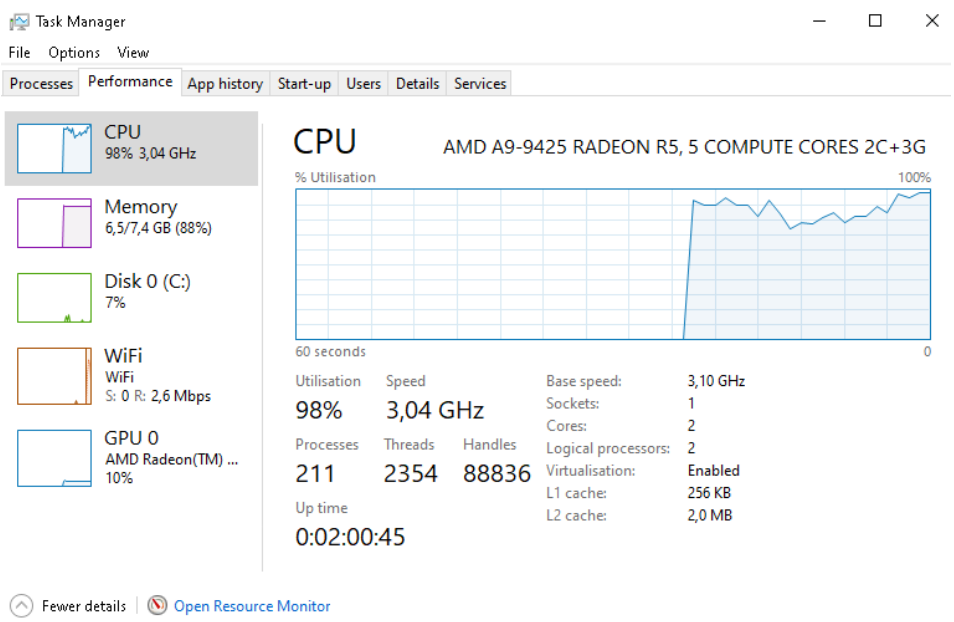


Figure 52 US-21 System performance during stress testing

US - 22 Trainer plans rehabilitation sessions

Table shows results after 2 tests:

Stress testing	50000
Load testing	1300

Gatling report after performing load testing:

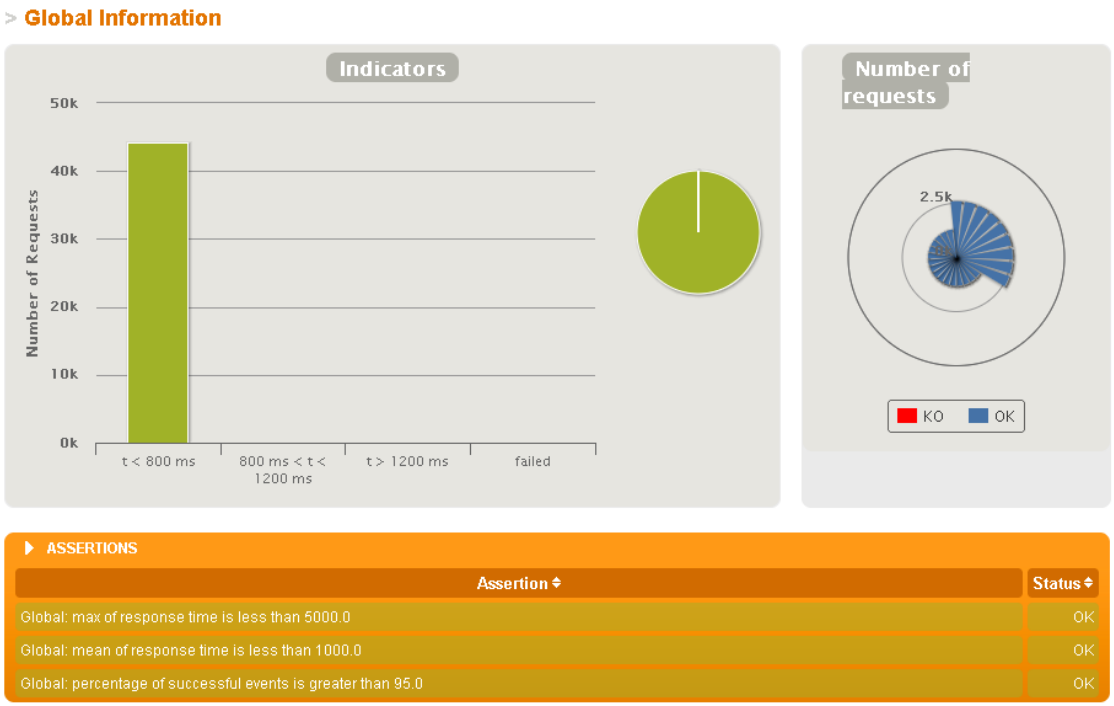


Figure 53 US-22 Testing overview

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	44200	44200	0	0%	172.656	0	4	7	93	295	3945	19	93
Home	2600	2600	0	0%	10.156	3	8	11	103	328	1659	30	117
Login	2600	2600	0	0%	10.156	1	4	7	211	453	3945	34	176
request_2	2600	2600	0	0%	10.156	0	2	5	45	161	3850	12	90
Logged	2600	2600	0	0%	10.156	1	5	8	112	291	3878	22	113
Logged Redirect 1	2600	2600	0	0%	10.156	0	2	4	22	110	1704	7	39
FindOwners	2600	2600	0	0%	10.156	1	5	8	158	421	3890	29	119
FindOwne...direct 1	2600	2600	0	0%	10.156	0	2	5	50	200	1701	14	88
OwnerInformation	2600	2600	0	0%	10.156	1	6	9	168	277	774	25	62
OwnerInf...direct 1	2600	2600	0	0%	10.156	0	3	5	70	327	1866	18	95
NewRehab2	1300	1300	0	0%	5.078	2	6	9	128	299	1118	24	82
NewRehab...direct 1	1300	1300	0	0%	5.078	0	3	6	41	201	697	11	48
NewRehab	1300	1300	0	0%	5.078	1	6	9	132	308	1083	23	70
NewRehab Redirect 1	1300	1300	0	0%	5.078	0	3	6	40	164	422	10	32
AddedNewRehab2	1300	1300	0	0%	5.078	1	6	9	169	405	3848	33	147
AddedNew...direct 1	1300	1300	0	0%	5.078	0	3	5	63	175	1699	12	56
AddedNewRehab	1300	1300	0	0%	5.078	1	6	9	145	461	3893	32	139
AddedNew...direct 1	1300	1300	0	0%	5.078	0	3	5	59	165	1703	11	55

Figure 54 US-22 Testing steps

System performance during stress testing:

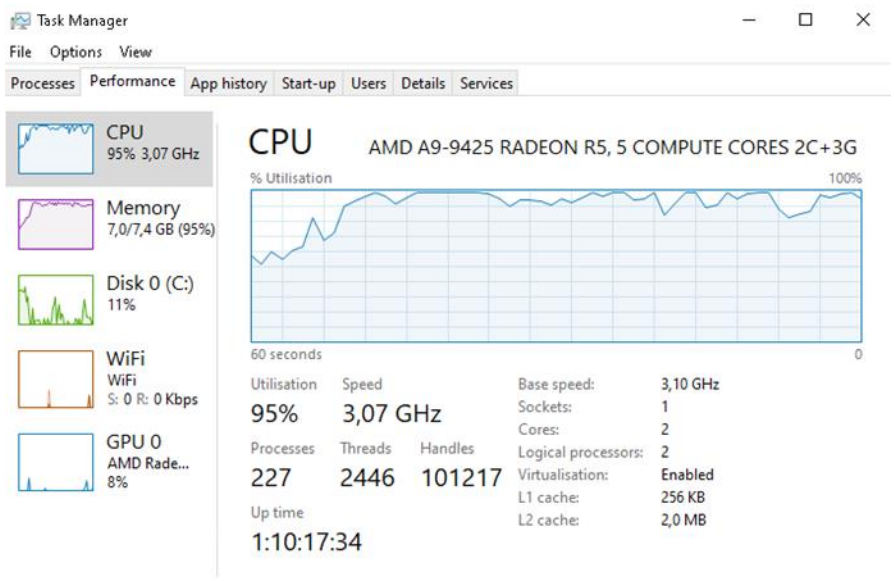


Figure 55 US-22 System performance during stress testing

US – 023 Owner sees rehabilitation sessions

Table shows results after 2 tests:

Stress testing	70000
Load testing	1900

Gatling report after performing load testing:

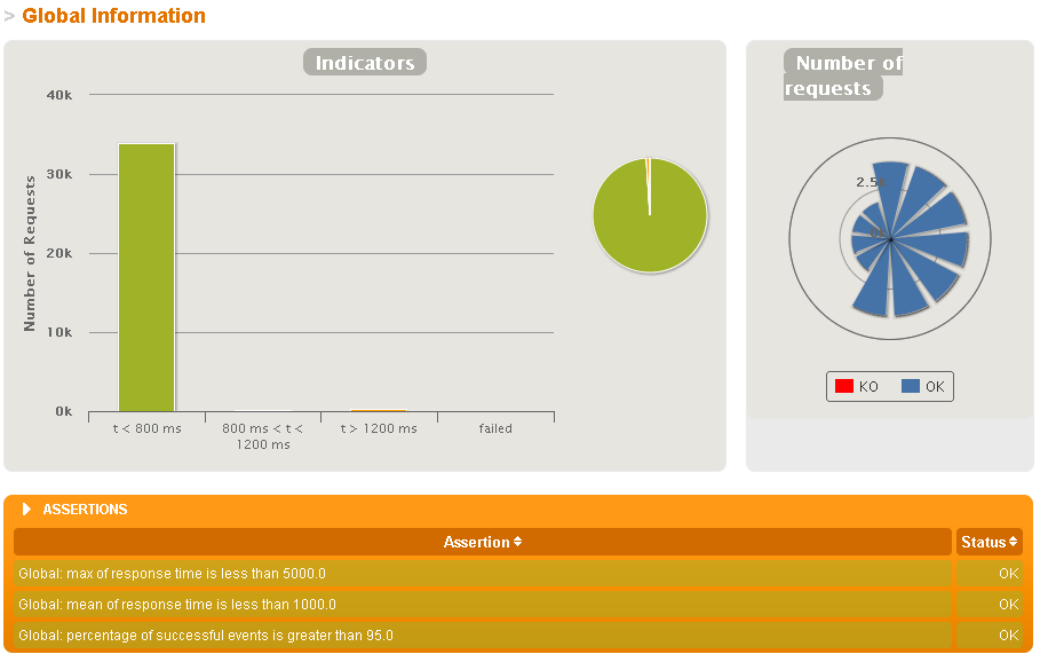
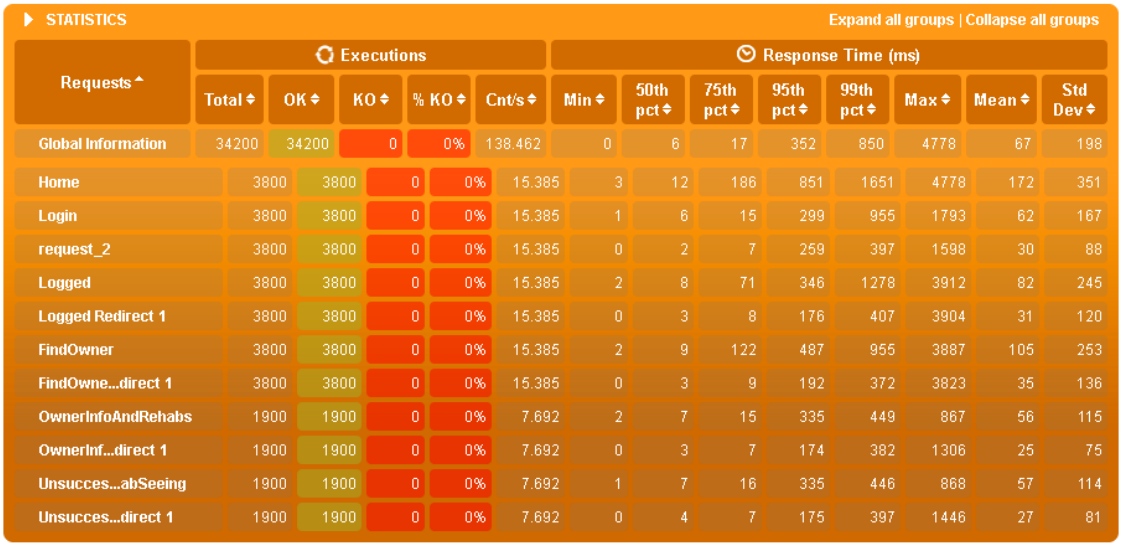


Figure 56 US-23 Testing overview



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	34200	34200	0	0%	138.462	0	6	17	352	850	4778	67	198
Home	3800	3800	0	0%	15.385	3	12	186	851	1651	4778	172	351
Login	3800	3800	0	0%	15.385	1	6	15	299	955	1793	62	167
request_2	3800	3800	0	0%	15.385	0	2	7	259	397	1598	30	88
Logged	3800	3800	0	0%	15.385	2	8	71	346	1278	3912	82	245
Logged Redirect 1	3800	3800	0	0%	15.385	0	3	8	176	407	3904	31	120
FindOwner	3800	3800	0	0%	15.385	2	9	122	487	955	3887	105	253
FindOwne...direct 1	3800	3800	0	0%	15.385	0	3	9	192	372	3823	35	136
OwnerInfoAndRehabs	1900	1900	0	0%	7.692	2	7	15	335	449	867	56	115
OwnerInf...direct 1	1900	1900	0	0%	7.692	0	3	7	174	382	1306	25	75
Unsucces...abSeeing	1900	1900	0	0%	7.692	1	7	16	335	446	868	57	114
Unsucces...direct 1	1900	1900	0	0%	7.692	0	4	7	175	397	1446	27	81

Figure 57 US-23 Testing steps

System performance during stress testing:

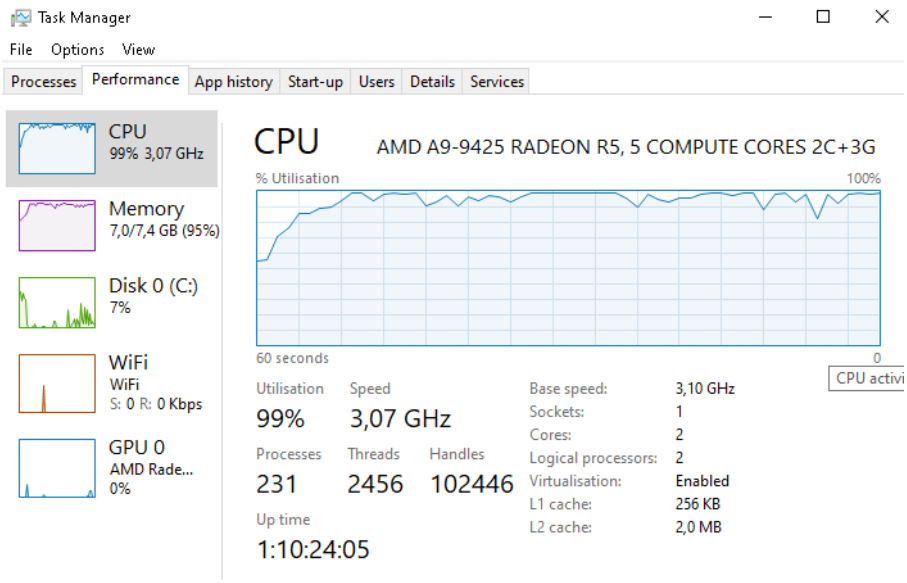


Figure 58 US-23 System performance during stress test

US – 024 Owner sees trainer's personal information

Table shows results after 2 tests:

Stress testing	50000
Load testing	800

Gatling report after performing load testing:

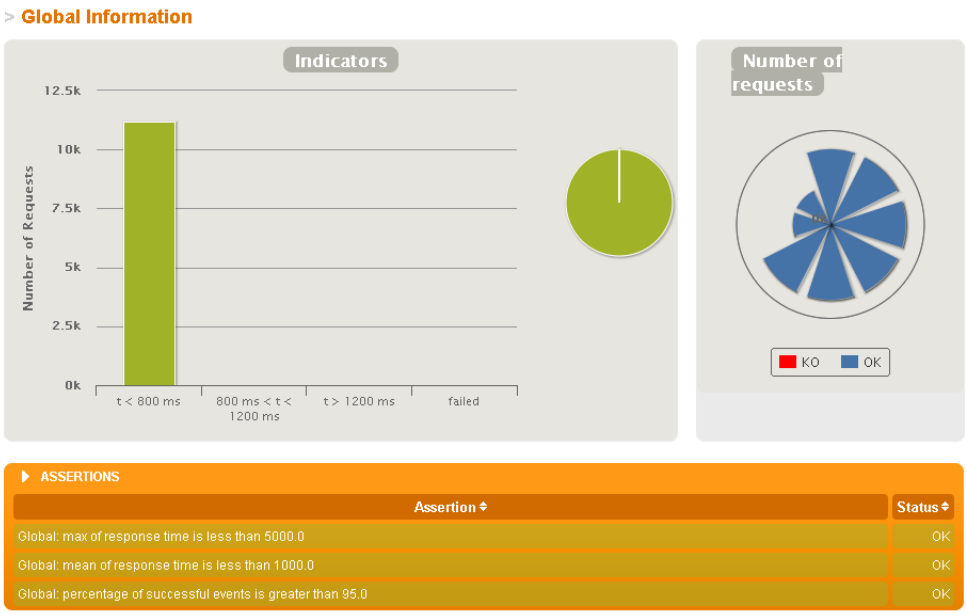


Figure 59 US-24 Testing overview

Figure 60 US-24 Testing steps

System performance during stress testing:

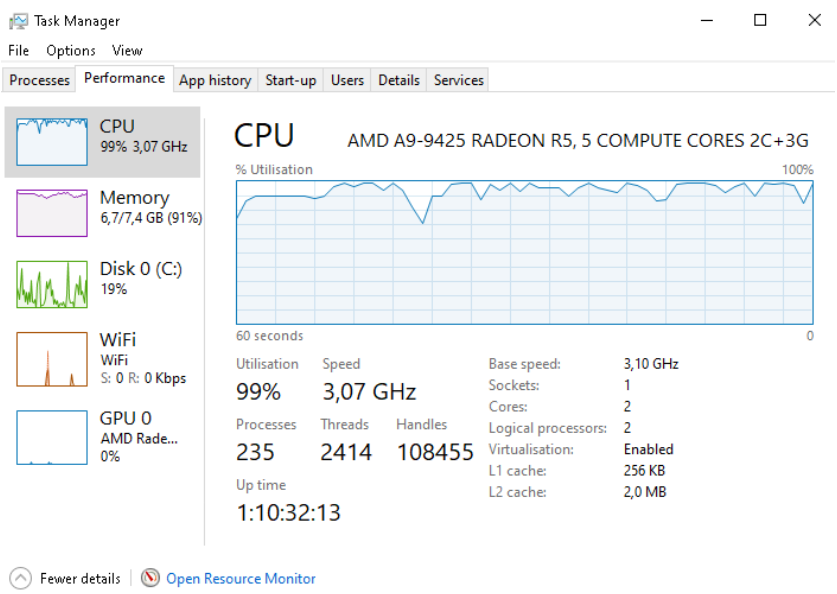


Figure 61 US-24 System performance during stress test

Conclusion

US# and description	Street test	Load test
US 018: Vet sees pet's visits	60000	2000
US 021: Training session organization	45000	1200
US 022: Trainer plans rehabilitation sessions	50000	1300
US 023: Owner sees rehabilitation sessions	70000	1900
US 024: Owner sees trainer's personal information	50000	800

As we can see, the problem will be located in both CPU and RAM, so both would need to be change in order to fix the bottleneck.

Manuel

The user stories of the project assigned to me were:

US (id)	Name
US-01	Vet adds a new medicine
US-02	Vet lists medicines
US-07	Vet prescribes medicines to a pet
US-08	Pet type's medicine checking
US-12	Adoption procedure

As the US-007 and US-008 are indivisible, I decided to do the performance testing of both at the same time due to US-008 is a business rule of the US-007.

Keeping this in mind, I have run 4 stress tests and 4 load tests (one of each per user story).

PC specs

Processor

- Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz

Memory

- Max Supported Size 16 GB

US-001 Vet adds a new medicine

In this snapshot we can see the system performance (CPU + RAM) at the end of the execution of the stress test with 80000 concurrent users:

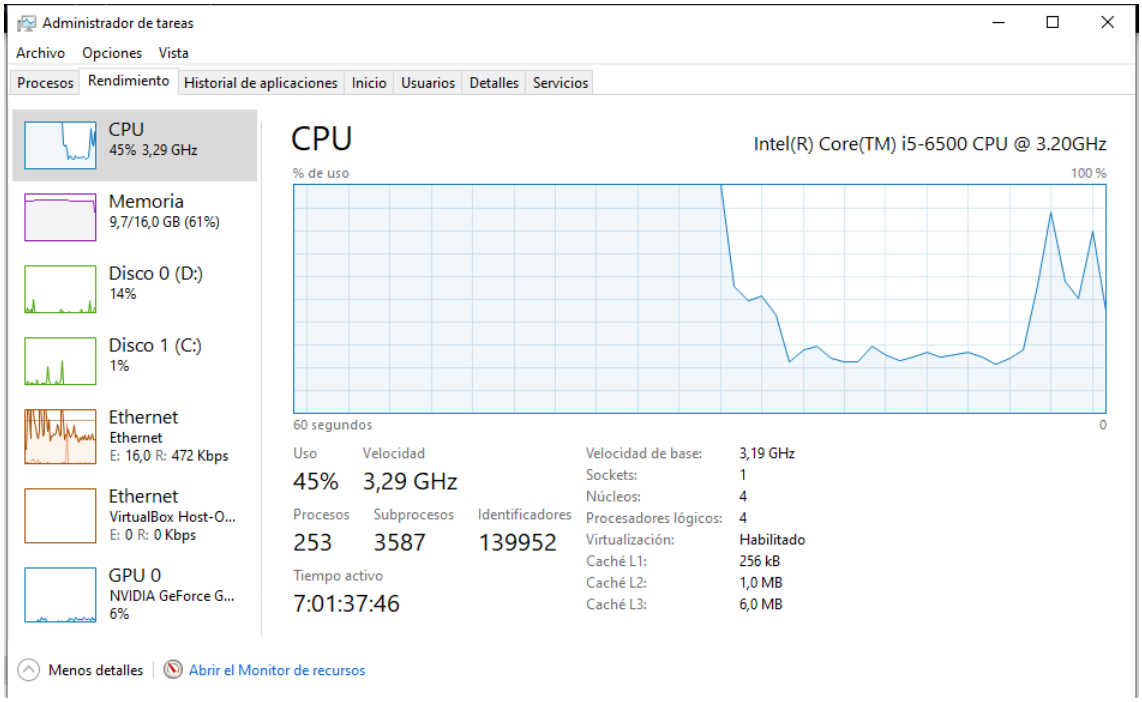


Figure 62 US-01 Stress test performance

And here we can see the Gatling report for this test:

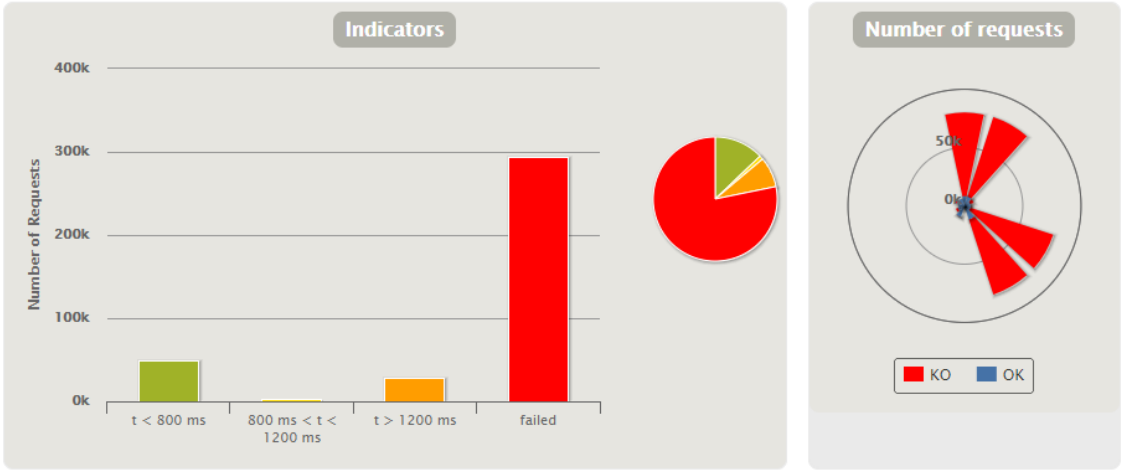


Figure 63 US-01 Stress test global

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	374314	80022	294292	79%	1290.738	0	2761	4498	17070	62747	86014	4957	9750
Home	80000	8192	71808	90%	275.862	173	4219	5053	13278	14801	62621	5847	4674
Login	80000	8192	71808	90%	275.862	0	328	2324	3400	4965	6615	1106	1310
logged	8192	4916	3276	40%	28.248	2	4111	5013	19113	24652	24840	4737	5822
logged Redirect 1	4916	4916	0	0%	16.952	1	14	1785	15697	20872	22016	2566	5212
MedicineListing	80000	5183	74817	94%	275.862	0	3727	5413	21050	78774	86014	7429	14142
MedicineForm	80000	12276	67724	85%	275.862	0	2884	5239	22883	76877	85995	6660	11433
Medicine...direct 1	2685	2685	0	0%	9.259	0	273	460	652	28802	28814	1162	4963
Medicine...direct 1	10055	9926	129	1%	34.672	0	116	343	54612	60002	71762	6346	16624
Medicine...rMessage	7783	5751	2032	26%	26.838	0	361	2111	3566	51798	81949	2069	7891
Medicine...direct 1	5094	5094	0	0%	17.566	0	12	178	581	47134	62634	1278	7065
MedicineCreated	9143	6446	2697	29%	31.528	1	753	2195	4176	51794	81950	2328	7904
Medicine...direct 1	6446	6445	1	0%	22.228	0	17	232	7108	47707	62719	1554	7326

Figure 64 US-01 Stress test details

Regarding to the load test, we can see in this charts that with 2000 users in 100 seconds, approximately the 90% of the petitions were resolved in less than 800ms:

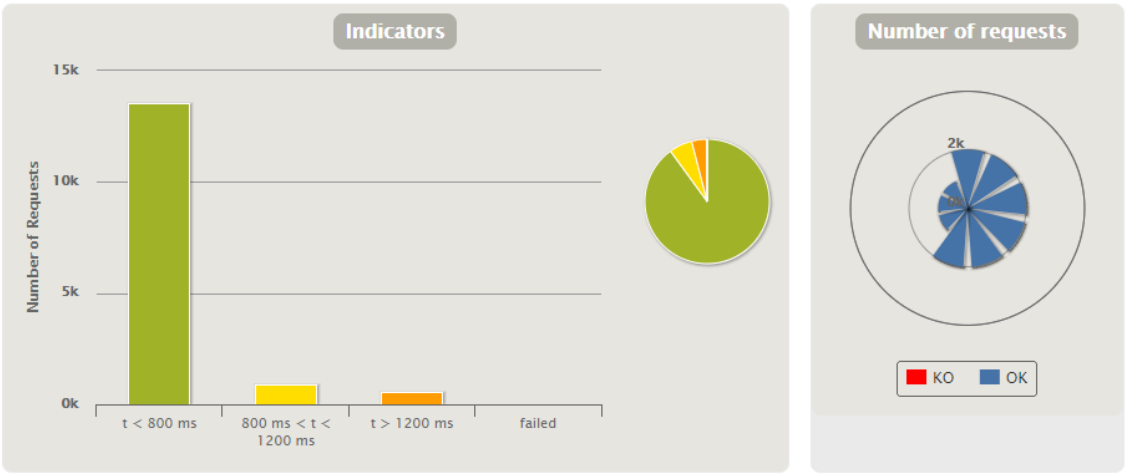


Figure 65 US-01 Load test global

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	15000	15000	0	0%	70.093	0	5	331	1075	1710	4199	232	399
Home	2000	2000	0	0%	9.346	1	2	3	272	999	2222	47	184
Login	2000	2000	0	0%	9.346	0	1	6	272	716	1546	48	146
logged	2000	2000	0	0%	9.346	1	157	504	1145	1664	3032	313	407
logged Redirect 1	2000	2000	0	0%	9.346	0	2	6	153	757	1534	30	113
MedicineListing	2000	2000	0	0%	9.346	82	629	954	1606	2210	4199	714	479
MedicineForm	2000	2000	0	0%	9.346	2	124	514	1144	1779	3072	309	420
Medicine...rMessage	1000	1000	0	0%	4.673	3	5	308	1019	1687	2407	219	375
MedicineCreated	1000	1000	0	0%	4.673	3	5	337	1090	1733	3058	237	409
Medicine...direct 1	1000	1000	0	0%	4.673	2	4	13	652	1449	2851	102	286

Figure 66 US-01 Load test details

US-002 Vet lists medicines

In this snapshot we can see the system performance (CPU + RAM) at the end of the execution of the stress test with 90000 concurrent users:

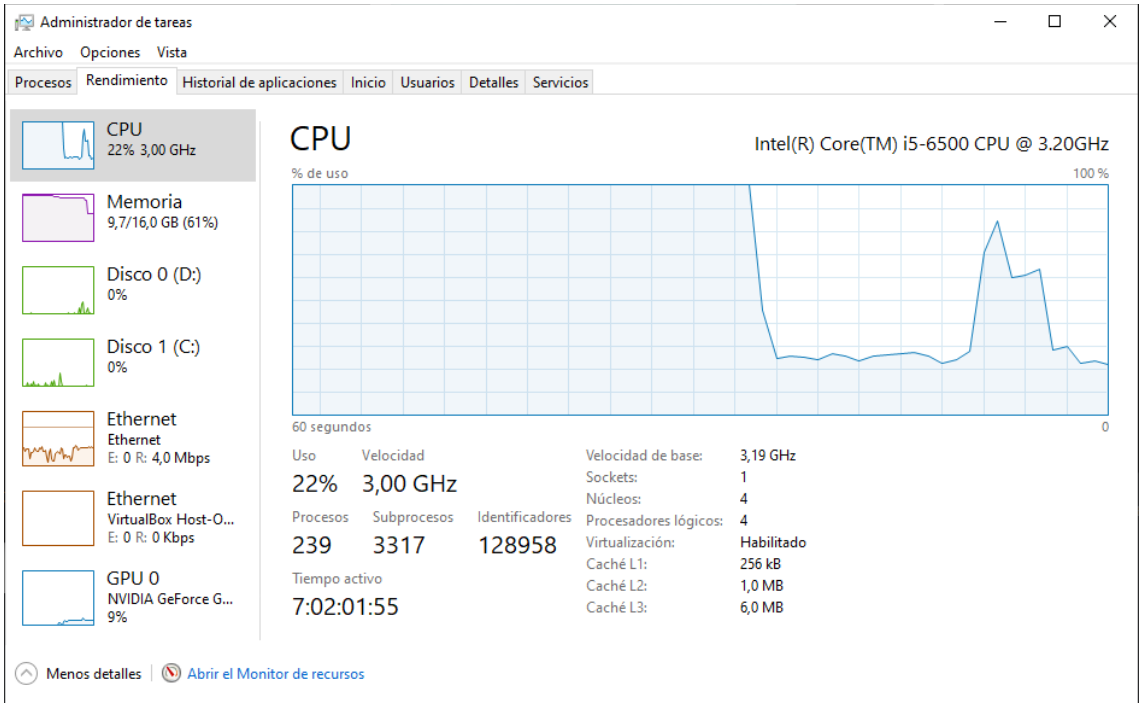


Figure 67 US-02 Stress test performance

And here we can see the Gatling report for this test:

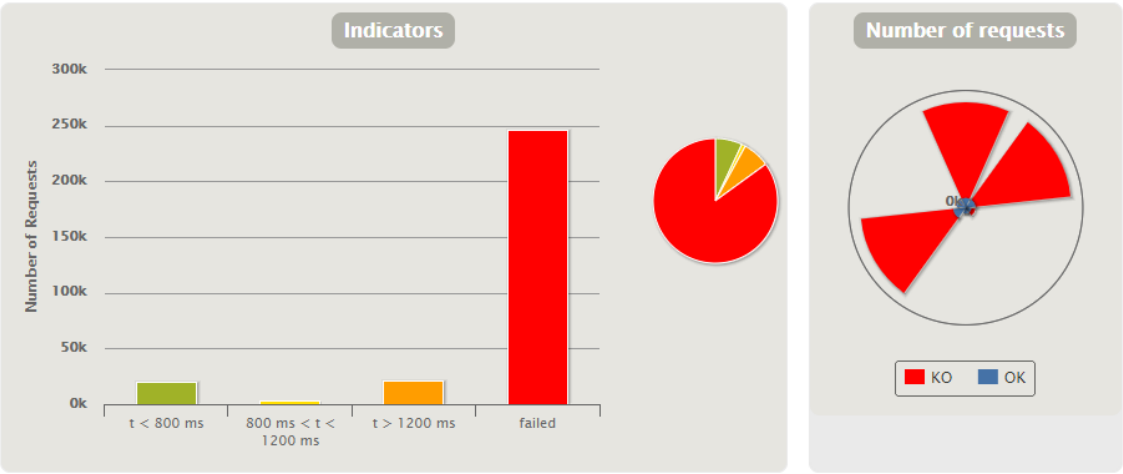


Figure 68 US-02 Stress test global

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	290417	43934	246483	85%	1357.089	0	2662	5232	24427	47044	83950	5192	8588
Home	90000	8192	81808	91%	420.561	202	5118	6360	13335	14331	63042	6457	4098
Login	90000	8196	81804	91%	420.561	0	454	2142	3184	4107	48600	1037	1216
logged	8196	4729	3467	42%	38.299	0	129	9692	44982	51913	60306	8969	15261
logged Redirect 1	4729	4729	0	0%	22.098	1	7	26	4673	31129	58872	1412	5782
MedicineListing	90000	10596	79404	88%	420.561	0	2313	8758	35973	57963	83950	7988	12681
Medicine...direct 1	7492	7492	0	0%	35.009	0	1477	4447	27659	46160	62666	4553	9291

Figure 69 US-02 Stress test details

Regarding to the load test, we can see in this charts that with 2200 users in 100 seconds (the left side of the interval), the 100% of the petitions were resolved in less than 800ms:

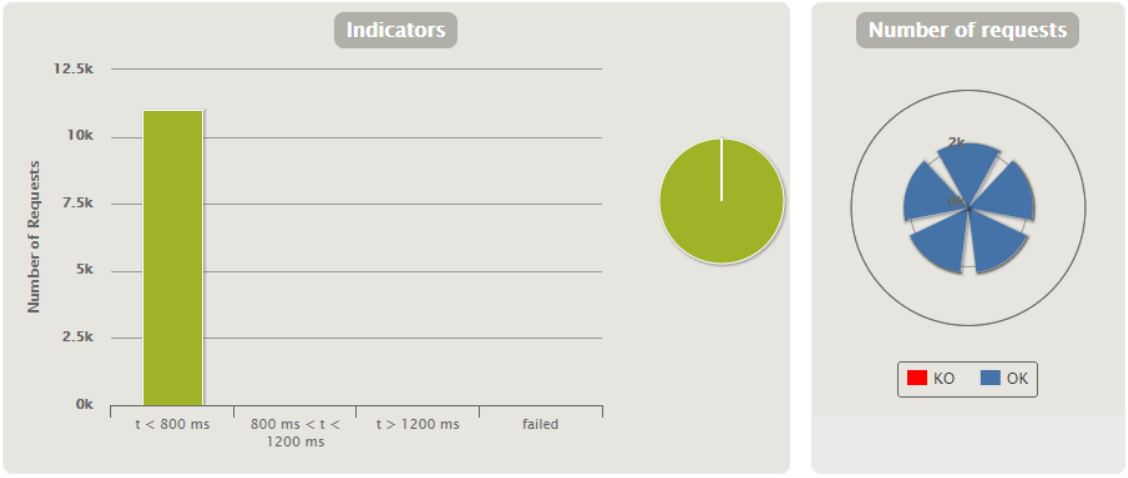


Figure 70 US-02 Load test global (left)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	11000	11000	0	0%	62.147	0	2	9	104	216	658	23	49
Home	2200	2200	0	0%	12.429	1	2	2	4	40	340	4	18
Login	2200	2200	0	0%	12.429	0	1	1	12	30	549	4	22
logged	2200	2200	0	0%	12.429	0	2	3	14	66	490	6	24
logged Redirect 1	2200	2200	0	0%	12.429	0	1	2	4	17	251	2	8
MedicineListing	2200	2200	0	0%	12.429	65	81	100	189	377	658	100	54

Figure 71 US-03 Load test details (left)

In the next ones, we can see that with 2500 users in 100 seconds (the right side of the interval), the 97% of the petitions were resolved in less than 800ms:

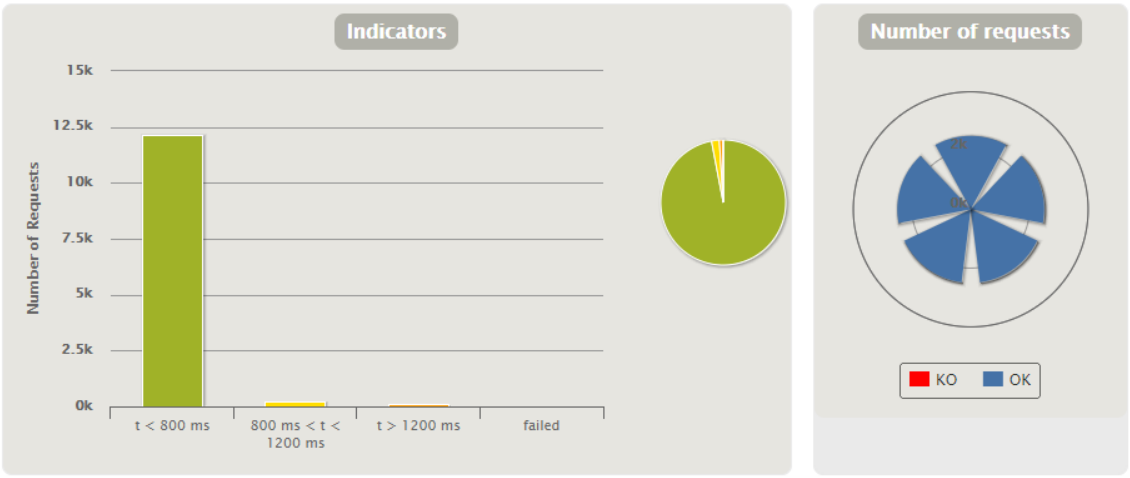


Figure 72 US-02 Load test global (right)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	12500	12500	0	0%	70.621	0	2	139	609	1254	6442	121	273
Home	2500	2500	0	0%	14.124	1	2	3	200	606	994	29	107
Login	2500	2500	0	0%	14.124	0	1	2	192	465	952	26	85
logged	2500	2500	0	0%	14.124	0	3	166	718	1684	6442	155	363
logged Redirect 1	2500	2500	0	0%	14.124	0	2	3	84	267	873	15	56
MedicineListing	2500	2500	0	0%	14.124	65	272	469	999	1709	4506	380	348

Figure 73 US-02 Load test details (right)

97% of the petitions resolved in less than 800ms is a more than acceptable percentage, but at this point, the response time of some petitions started to increase fast, so I kept this value as the end of the interval.

US-007 & US-008 Vet prescribes medicines to a pet

In this snapshot we can see the system performance (CPU + RAM) at the end of the execution of the stress test with 80000 concurrent users:

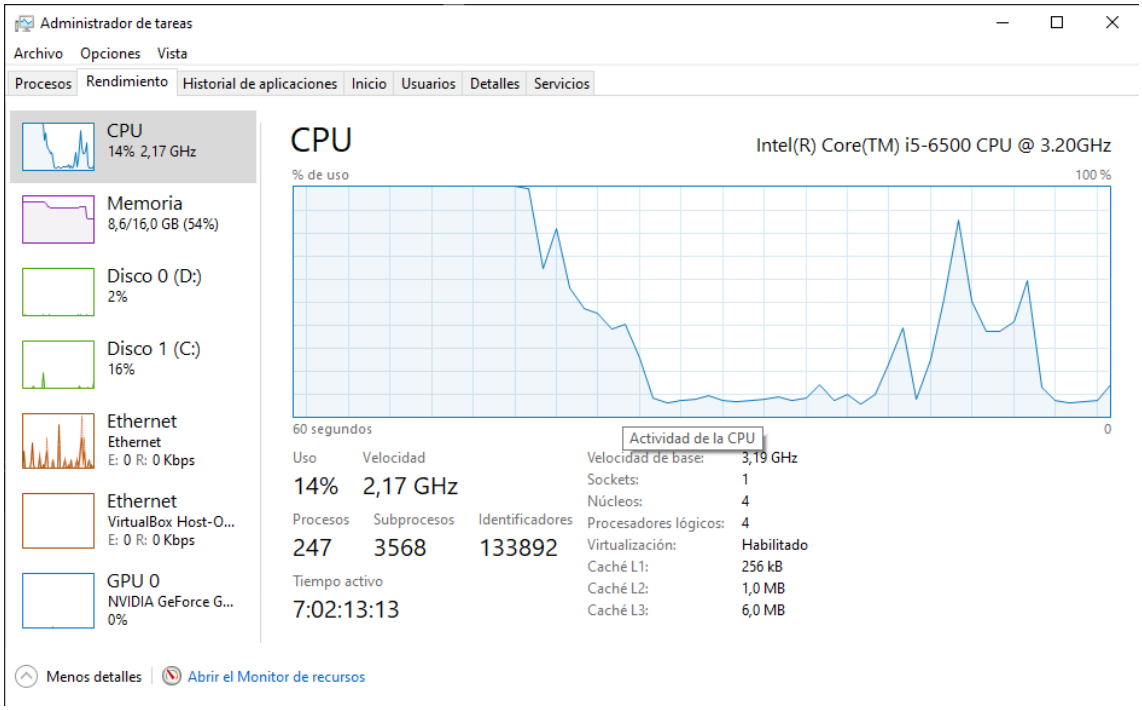


Figure 74 US-07&8 Stress test performance

And here we can see the Gatling report for this test:

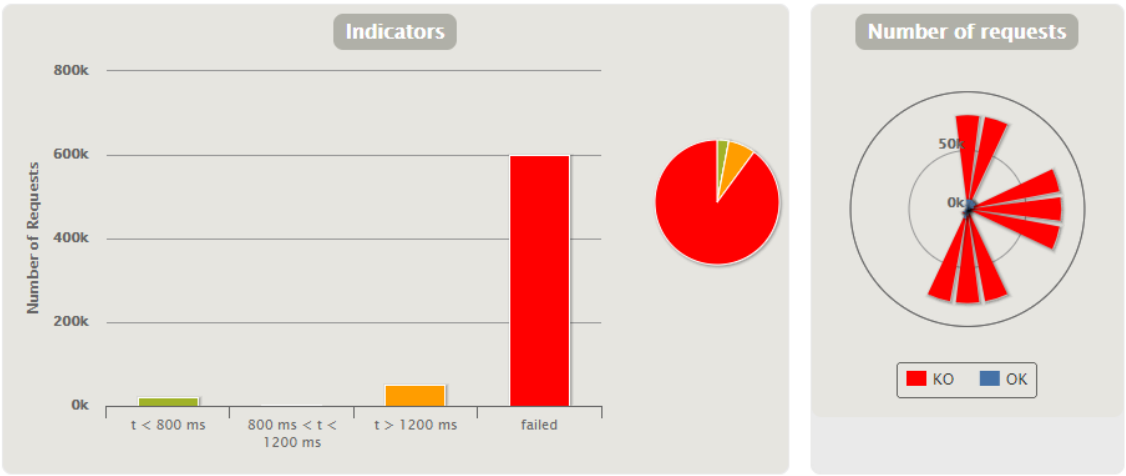


Figure 75 US-07&8 Stress test global

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	673608	74438	599170	89%	1358.081	0	2328	3431	30547	60001	62923	4929	9868
Home	80000	8192	71808	90%	161.29	326	5372	7109	10931	11615	61649	6195	2806
Login	80000	8192	71808	90%	161.29	0	461	1409	3661	3994	4599	1055	1264
logged	8192	7996	196	2%	16.516	1	7	1035	5577	10028	43166	1295	3888
logged Redirect 1	7996	7932	64	1%	16.121	1	8	15	2406	4229	60003	853	5527
OwnerList	80000	5291	74709	93%	161.29	0	2371	3324	39323	62118	62923	5880	12557
OwnerListing	80000	2140	77860	97%	161.29	0	2530	3399	49196	60719	62326	6166	13159
OwnerSelection	80000	2153	77847	97%	161.29	0	2243	2529	4846	56170	62821	3307	7666
OwnerLis...direct 1	19	19	0	0%	0.038	1	10	24925	25110	25257	25294	10517	12325
OwnerLis...direct 1	26	25	1	4%	0.052	1	25166	35826	40196	55051	60001	25929	13897
MedicalR...dListing	80000	3961	76039	95%	161.29	0	2266	2390	23503	38366	60007	3390	7799
MedicalR...election	80000	5181	74819	94%	161.29	0	2281	2392	29799	41007	60016	3995	8614
PrescriptionForm	80000	7942	72058	90%	161.29	0	2288	2388	34774	41339	60279	5287	10255
Prescrip...rMessage	4490	3662	828	18%	9.052	0	28361	37962	43414	60000	60002	27480	13115
PrescriptionCreated	5281	4187	1094	21%	10.647	0	31473	37151	45568	60000	60007	27275	14288
Prescrip...direct 1	1639	1639	0	0%	3.304	0	25206	26779	38504	41085	41418	26412	7592
MedicalR...direct 1	611	611	0	0%	1.232	0	25221	36262	38256	39223	40062	27871	8112
MedicalR...direct 1	265	265	0	0%	0.534	0	23468	25690	35416	37527	41102	24332	5458
OwnerSel...direct 1	55	55	0	0%	0.111	3	25276	39617	40111	40889	41215	29406	8804
Prescrip...direct 1	851	851	0	0%	1.716	0	37552	38935	39776	40077	40185	29965	12851
Prescrip...direct 1	4183	4144	39	1%	8.433	0	31671	38845	42157	58504	60003	28401	13080

Figure 76 US-07&8 Stress test details

Regarding to the load test, we can see in this charts that with 2000 users in 100 seconds (the left side of the interval), the 100% of the petitions were resolved in less than 800ms:

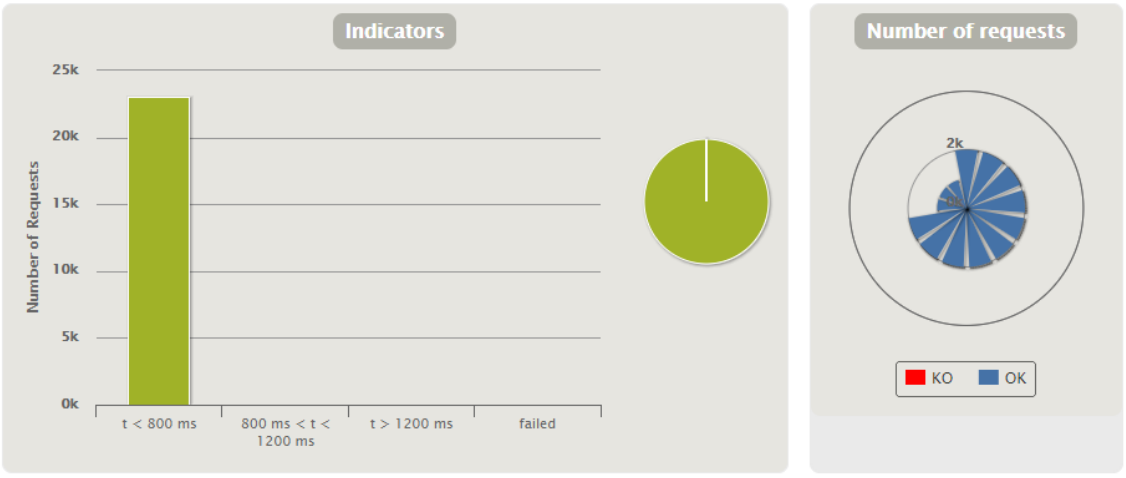


Figure 77 US-07&8 Load test global (left)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	23000	23000	0	0%	90.909	0	4	10	35	43	127	8	10
Home	2000	2000	0	0%	7.905	1	2	2	3	6	23	2	1
Login	2000	2000	0	0%	7.905	0	1	1	2	3	8	1	1
logged	2000	2000	0	0%	7.905	1	2	2	4	6	12	2	1
logged Redirect 1	2000	2000	0	0%	7.905	0	2	2	4	6	14	2	1
OwnerList	2000	2000	0	0%	7.905	1	2	3	5	8	22	2	2
OwnerListing	2000	2000	0	0%	7.905	6	11	15	24	52	107	13	8
OwnerSelection	2000	2000	0	0%	7.905	5	9	12	22	49	127	11	8
MedicalR...dListing	2000	2000	0	0%	7.905	2	5	7	14	31	83	6	5
MedicalR...election	2000	2000	0	0%	7.905	7	12	14	22	42	88	13	6
PrescriptionForm	2000	2000	0	0%	7.905	3	5	7	17	32	77	7	6
Prescrip...rMessage	1000	1000	0	0%	3.953	3	5	6	14	32	107	7	6
PrescriptionCreated	1000	1000	0	0%	3.953	35	39	41	47	60	98	40	5
Prescrip...direct 1	1000	1000	0	0%	3.953	7	12	14	17	26	92	13	4

Figure 78 US-07&8 Load test details (left)

In the next ones, we can see that with 2250 users in 100 seconds (the right side of the interval), the 74% of the petitions were resolved in less than 800ms:

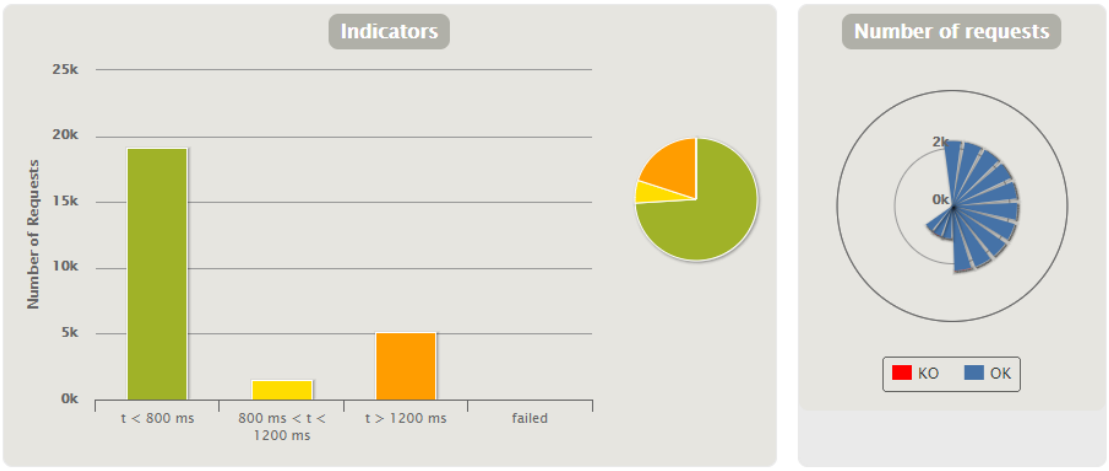


Figure 79 US-07&8 Load test global (right)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	25870	25844	26	0%	90.455	0	10	863	2444	16331	57844	881	2921
Home	2250	2250	0	0%	7.867	1	2	2	4	11	126	2	4
Login	2250	2250	0	0%	7.867	0	1	2	3	8	132	1	5
logged	2250	2250	0	0%	7.867	1	2	3	74	2203	30473	122	1276
logged Redirect 1	2250	2250	0	0%	7.867	0	2	3	6	531	1920	15	108
OwnerList	2250	2250	0	0%	7.867	1	2	3	800	1848	2212	100	347
OwnerListing	2250	2250	0	0%	7.867	6	18	466	2479	16986	33250	846	2883
OwnerSelection	2250	2250	0	0%	7.867	1	55	1568	4873	24101	32651	1413	3894
MedicalR...dListing	2250	2250	0	0%	7.867	1	698	1751	6471	21953	32637	1645	3834
MedicalR...election	2250	2250	0	0%	7.867	0	976	1836	6892	22436	32973	1811	3846
PrescriptionForm	2250	2237	13	1%	7.867	1	863	1718	6110	21986	32628	1663	3771
Prescrip...rMessage	1125	1123	2	0%	3.934	3	807	1755	6299	21118	32106	1641	3578
PrescriptionCreated	1125	1114	11	1%	3.934	1	863	1781	6864	23178	57844	1860	4553
Prescrip...direct 1	1108	1108	0	0%	3.874	0	623	1696	5945	20037	31986	1537	3368
OwnerLis...direct 1	2	2	0	0%	0.007	850	986	1053	1107	1118	1121	986	136
OwnerLis...direct 1	2	2	0	0%	0.007	34	355	516	644	670	676	355	321
OwnerSel...direct 1	2	2	0	0%	0.007	0	1	1	1	1	1	1	1
MedicalR...direct 1	2	2	0	0%	0.007	0	0	0	0	0	0	0	0
MedicalR...direct 1	2	2	0	0%	0.007	1	1	1	1	1	1	1	0
Prescrip...direct 1	2	2	0	0%	0.007	0	1	1	1	1	1	1	1

Figure 80 US-07&8 Load test details (right)

74% of the petitions resolved in less than 800ms is a relative acceptable percentage, but at this point, the response time of some petitions started to increase fast, so I kept this value as the end of the interval.

Regardless of this, I would choose a value near to the left side of the interval because of a fast increase of the response time in few users.

US 12 Adoption procedure

In this snapshot we can see the system performance (CPU + RAM) at the end of the execution of the stress test with 85000 concurrent users:

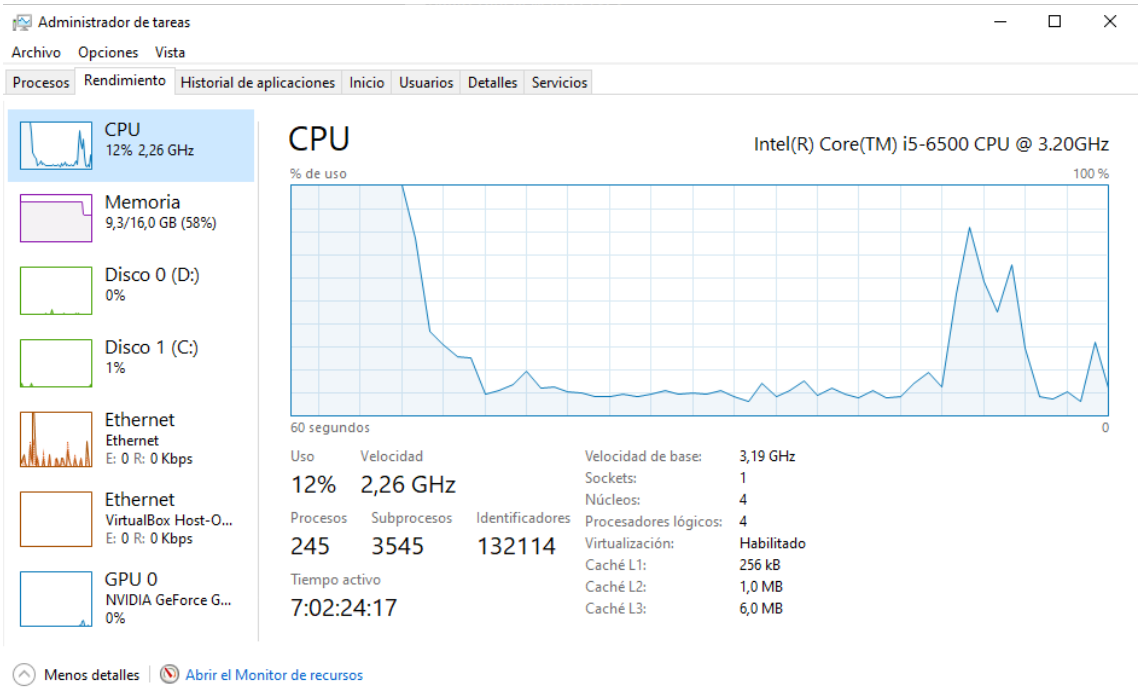


Figure 81 US-012 Stress test performance

And here we can see the Gatling report for this test

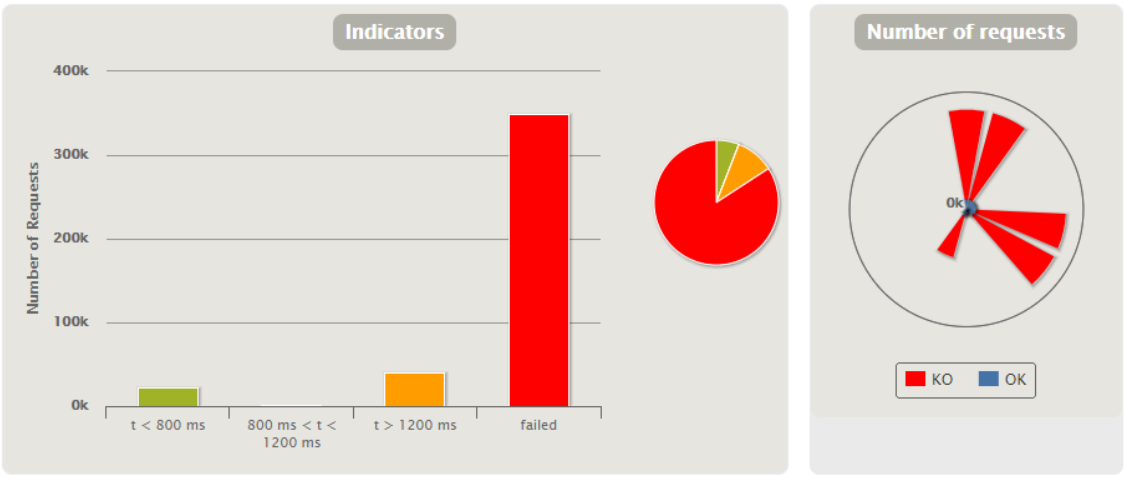


Figure 82 US-012 Stress test global

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	412229	63438	348791	85%	934.76	0	2317	3698	43001	53800	61225	5827	11603
Home	85000	8192	76808	90%	192.744	2894	4462	5143	11546	12730	61225	5533	3161
Login	85000	8192	76808	90%	192.744	0	293	2069	2851	3200	3458	862	1059
logged	8192	8192	0	0%	18.576	1	18	831	18018	32330	54480	2673	6797
logged Redirect 1	8192	8139	53	1%	18.576	1	26	295	49204	55171	60002	6298	15407
ListHomelessPets	85000	7899	77101	91%	192.744	0	2302	2391	24569	48202	60518	4358	8462
PetAdoptionForm	85000	6479	78521	92%	192.744	0	2297	2385	48614	60001	60635	6457	13460
PetAdopt...rMessage	3401	2572	829	24%	7.712	41	44157	47903	57471	60002	60014	40128	15790
PetAdopted	4850	3647	1203	25%	10.998	2003	44388	48523	55767	60001	60006	38671	17163
AdoptionHistory	42500	5173	37327	88%	96.372	0	2284	2377	45792	57367	60009	7349	14040
PetAdopt...direct 1	3646	3505	141	4%	8.268	0	45834	47777	58510	60001	60024	45940	7472
PetAdopt...direct 1	139	139	0	0%	0.315	0	44102	49782	50111	50130	50153	43764	11188
Adoption...direct 1	1158	1158	0	0%	2.626	0	47897	49744	50108	50140	50164	43712	14043
ListHome...direct 1	81	81	0	0%	0.184	0	49002	49485	49684	50101	50112	44496	13710
PetAdopt...direct 1	70	70	0	0%	0.159	0	11568	43478	43803	44090	44100	21196	19738

Figure 83 US-012 Stress test details

Regarding to the load test, we can see in this charts that with 3200 users in 100 seconds (the left side of the interval), the 100% of the petitions were resolved in less than 800ms:

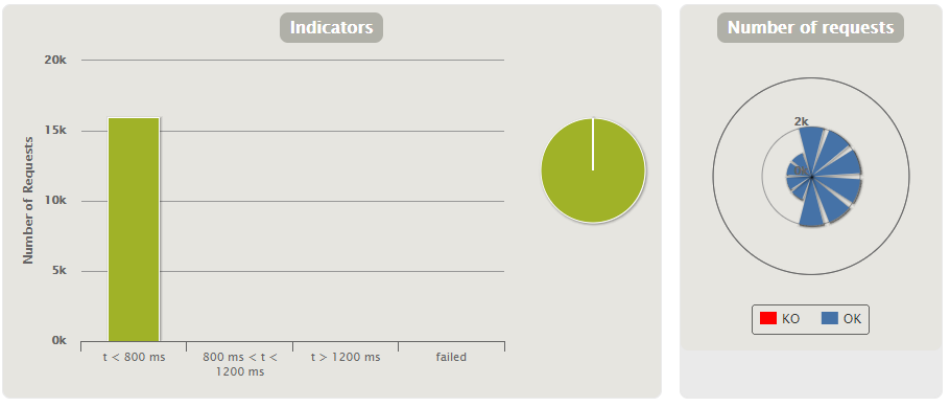


Figure 84 US-012 Load test global (left)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	16000	16000	0	0%	68.085	0	3	11	47	55	202	8	13
Home	2000	2000	0	0%	8.511	1	2	2	4	10	28	2	2
Login	2000	2000	0	0%	8.511	0	1	1	2	4	12	1	1
logged	2000	2000	0	0%	8.511	0	2	2	4	5	9	2	1
logged Redirect 1	2000	2000	0	0%	8.511	0	2	2	3	5	10	2	1
ListHomelessPets	2000	2000	0	0%	8.511	2	3	4	7	16	158	4	5
PetAdoptionForm	2000	2000	0	0%	8.511	5	8	11	18	39	198	10	7
PetAdopt...rMessage	1000	1000	0	0%	4.255	9	13	15	22	38	202	15	8
PetAdopted	1000	1000	0	0%	4.255	12	15	18	24	43	189	17	8
PetAdopt...direct 1	1000	1000	0	0%	4.255	7	11	12	17	29	62	11	4
AdoptionHistory	1000	1000	0	0%	4.255	44	50	53	57	65	133	51	6

Figure 85 US-012 Load test details (left)

In the next ones, we can see that with 3000 users in 100 seconds (the right side of the interval), the 88% of the petitions were resolved in less than 800ms:

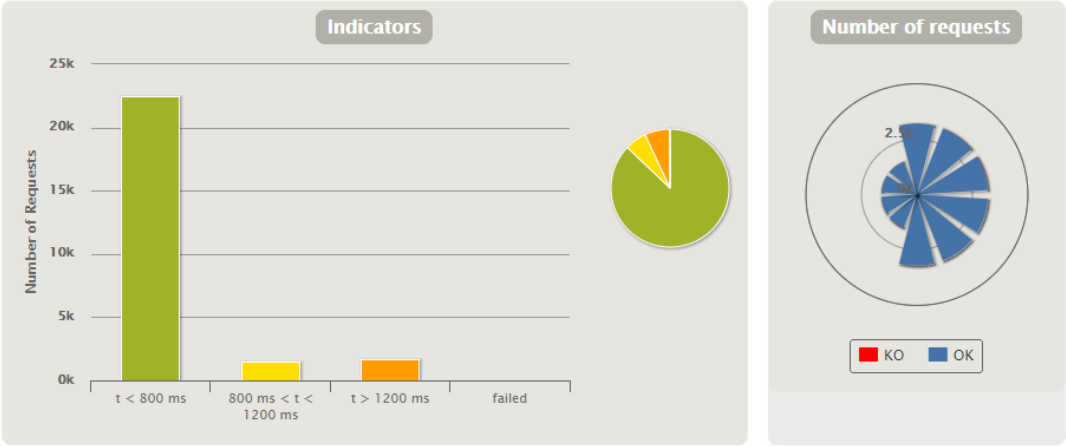


Figure 86 US-012 Load test global (right)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	25600	25600	0	0%	106.667	0	11	479	1420	8003	29377	460	1434
Home	3200	3200	0	0%	13.333	1	2	3	4	6	87	3	2
Login	3200	3200	0	0%	13.333	0	1	2	398	629	1544	47	152
logged	3200	3200	0	0%	13.333	1	3	348	2763	9955	22392	544	1831
logged Redirect 1	3200	3200	0	0%	13.333	0	3	142	660	881	1667	124	240
ListHomelessPets	3200	3200	0	0%	13.333	2	7	566	2383	8744	29377	600	1736
PetAdoptionForm	3200	3200	0	0%	13.333	5	382	729	2672	9628	21820	761	1802
PetAdopt...rMessage	1600	1600	0	0%	6.667	9	413	753	3149	9323	21879	761	1661
PetAdopted	1600	1600	0	0%	6.667	13	440	779	2691	9773	20840	782	1677
PetAdopt...direct 1	1600	1600	0	0%	6.667	8	361	705	2918	10436	21571	780	1875
AdoptionHistory	1600	1600	0	0%	6.667	73	520	839	2801	9698	21186	878	1646

Figure 87 US-012 Load test details (right)

88% of the petitions resolved in less than 800ms is a more than acceptable percentage, but at this point, the response time of some petitions started to increase fast, so I kept this value as the end of the interval.

Conclusion

User story	Stress test	Load test (*) (**)
US-001 Vet adds a new medicine	80000	2000
US-002 Vet lists medicines	90000	2200 – 2500
US-007 & US-008 Vet prescribes medicines to a pet	80000	2000 – 2250
US-012 Adoption procedure	85000	3000 – 3200

(*) The number of users is loaded in a period of 100 seconds.

(**) When a range is given, it means that the maximum number of users with acceptable behavior is between the two numbers.

According to the results, the main bottleneck is the memory, so it would be needed to change the RAM in order to obtain better results and avoid Bottleneck.

Iván

During the duration of this project, I was in charge of implementing the following US:

US (id)	Name
US-03	Medical Record Creation
US-04	Owner sees pet's medical record
US-05	Trainers has access to medical records
US-06	Vet sees a pet's medical record
US-09	Vet changes a Pet's Medical Record

US 004, US 005 and US 006 works practically the same with the exception of the role which can have access to the listing, so in order to avoid meaningless repetitions, I merged the three of them into one performance test (from now on, it will be called US 00L). In the same vein, additional functions like a delete for medical record were implemented, so I decided to test that (from now on, it will be called US 00D)

So, the table which with I will be working looks as follows

US (id)	Name
US-03	Medical Record Creation
US-09	Vet changes a Pet's Medical Record
US-0L	Listing Medical Record
US-0D	Delete Medical Record

PC specs

HP Omen 15

Processor

- Intel i7-6700HQ

Memory

- Max Supported Size 8 GB

Performance Analysis

When creating the Performance Test I decided to focus in two different points, the Stress Point, where the application will fail almost completely, and the Load test, when the results, while not being horrible, will not be optimal. In order to select some criteria, I consider a Stress test correct if it had more than 90% of KO, and a Load test correct if its mean response time was between 1000 and 2000 ms.

I will show capture of both my system performance (CPU + RAM) and the Gatling report

US 003

- Stress Test
    - System Performance

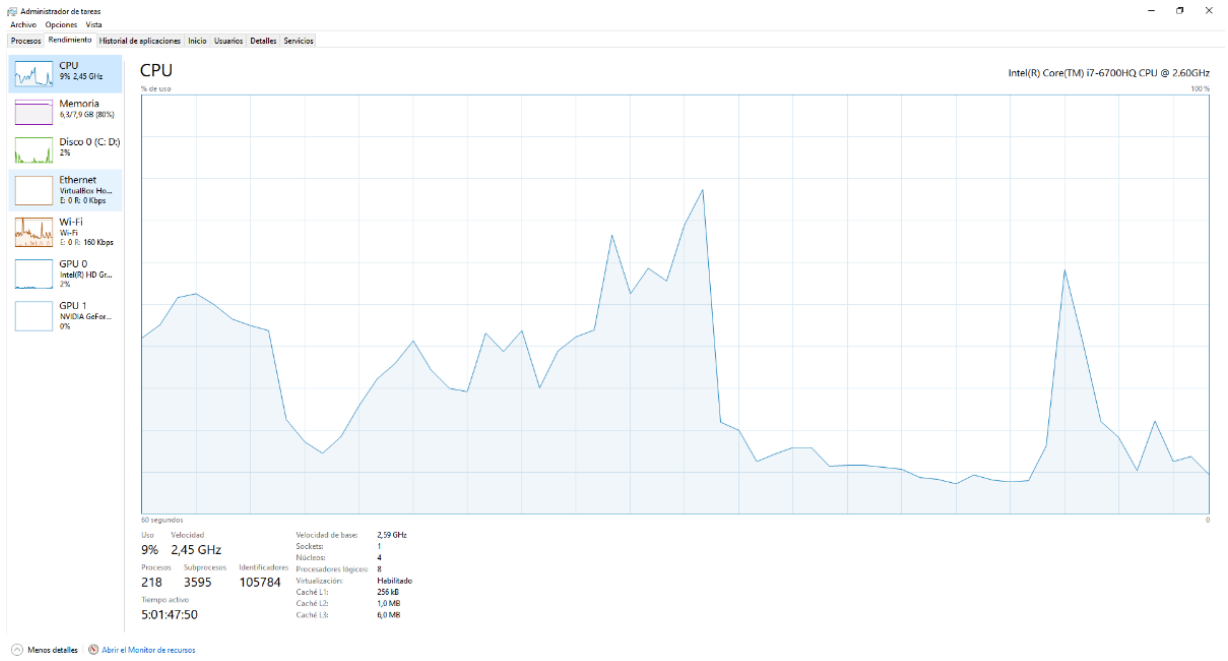
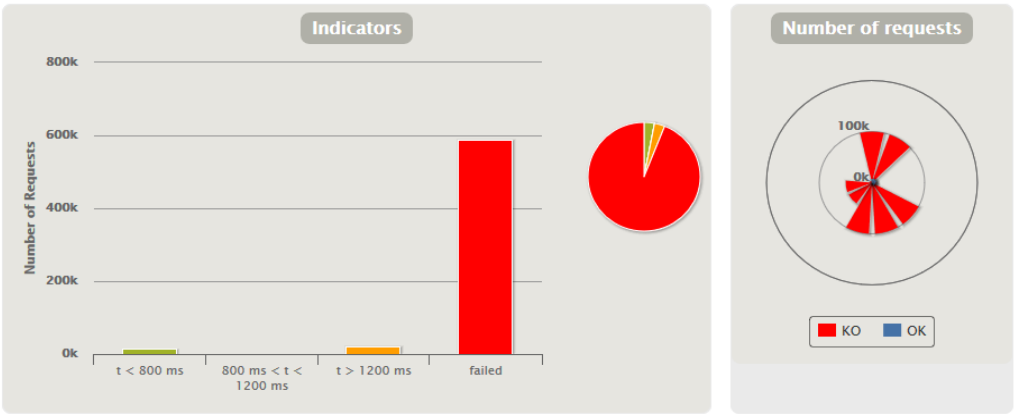


Figure 88 US-03 Stress test performance

- Gatling Report



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	624521	37055	587466	94%	1266.777	0	1899	4924	18854	60076	64658	5701	11084
home	100000	7417	92583	93%	202.84	14	13857	15903	19452	21464	22202	13995	3369
login	100000	8192	91808	92%	202.84	0	823	1752	3951	6282	61209	1361	2304
logged	8192	8137	55	1%	16.617	0	533	6371	15057	25162	62833	3873	7000
logged Redirect 1	8137	6876	1261	15%	16.505	1	369	8356	60001	60091	60770	12470	21783
findowners	100000	5974	94026	94%	202.84	0	1863	2944	7377	60002	60901	3281	7827
listowners	100000	410	99590	100%	202.84	0	826	1954	60011	61034	63732	5518	15354
ownerview	100000	49	99951	100%	202.84	0	1728	2450	9074	60002	64658	4593	12167
medicalr...negative	50000	0	50000	100%	101.42	0	2003	3795	7142	60002	63261	4531	11014
medicalr...positive	50000	0	50000	100%	101.42	0	2010	3869	8134	60009	63255	5136	12224
creation	4484	0	4484	100%	9.095	2000	2004	2014	60001	62373	63256	9023	19001
error	3708	0	3708	100%	7.521	2000	2004	2012	60001	62077	63255	7761	17383

Figure 89 US-03 Stress test gatling results

- Load Test
  - System Performance

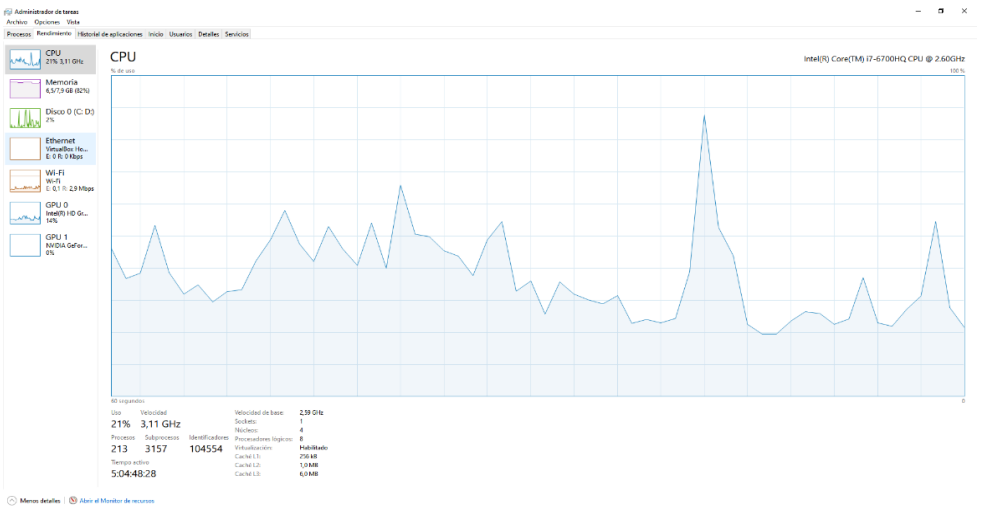
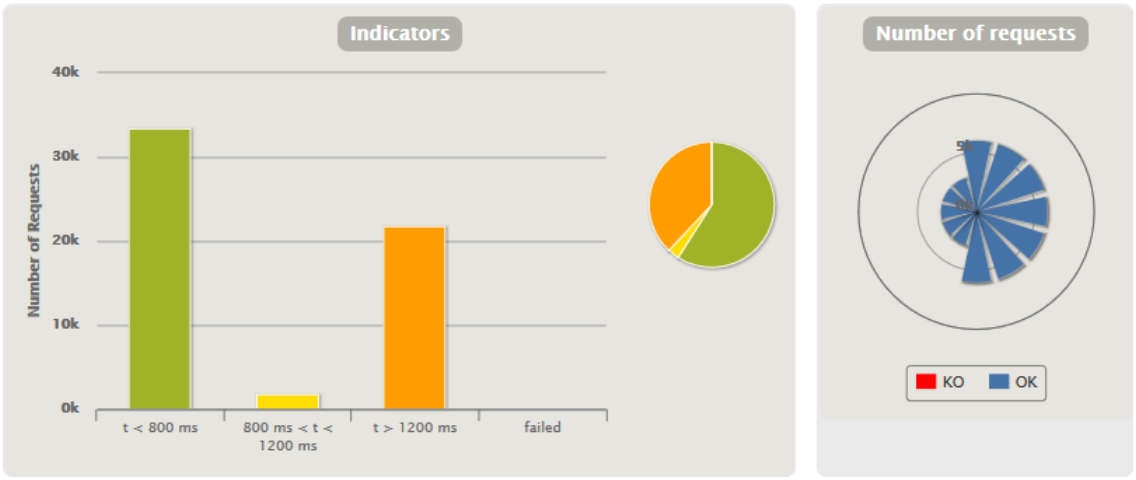


Figure 90 US-03 Load test performance

- Gatling Report



ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 1000.0	KO
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	57000	57000	0	0%	218.391	0	123	2437	4799	6536	15808	1324	1791
home	6000	6000	0	0%	22.989	1	4	5	9	15	104	5	4
login	6000	6000	0	0%	22.989	0	3	7	1549	1689	1759	227	490
logged	6000	6000	0	0%	22.989	0	123	1951	4117	6272	10325	1058	1545
logged Redirect 1	6000	6000	0	0%	22.989	0	5	1329	3905	4163	4226	734	1209
findowners	6000	6000	0	0%	22.989	1	471	2715	4042	4177	4243	1346	1541
listowners	6000	6000	0	0%	22.989	6	2021	3753	5495	7365	12162	2250	1892
ownerview	6000	6000	0	0%	22.989	2	2025	3693	5505	7347	15808	2205	1932
medicalr...negative	3000	3000	0	0%	11.494	2	1935	3674	5643	7291	11086	2151	1999
medicalr...positive	3000	3000	0	0%	11.494	2	1964	3704	5535	7415	12652	2162	2009
creation	3000	3000	0	0%	11.494	1	68	3592	5536	7376	11065	1769	2142
error	3000	3000	0	0%	11.494	2	64	3611	5512	7350	11359	1770	2138
creation Redirect 1	3000	3000	0	0%	11.494	1	39	3476	5466	7423	12536	1647	2079

Figure 91 US-03 Load test gatling results

US 009

- Stress Test
  - System Performance

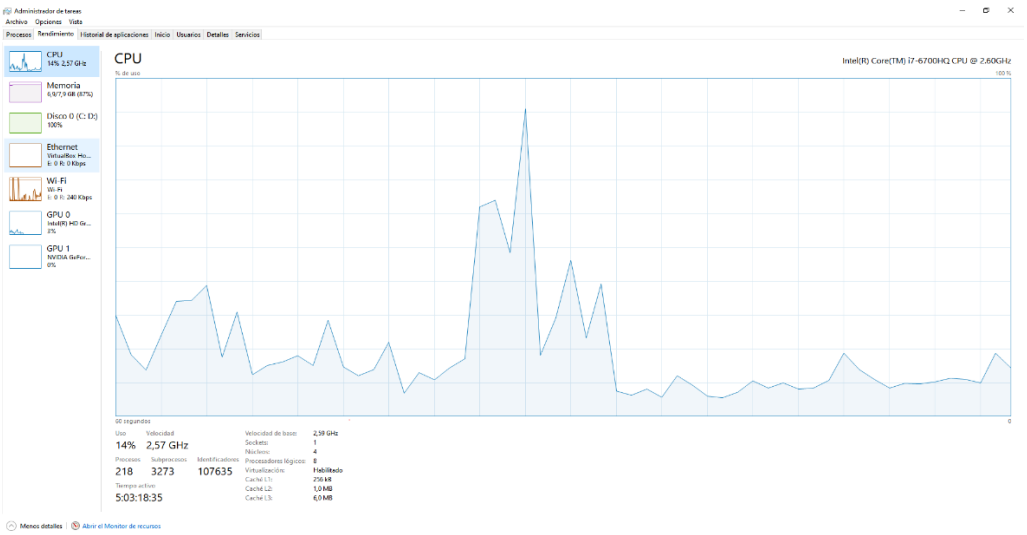
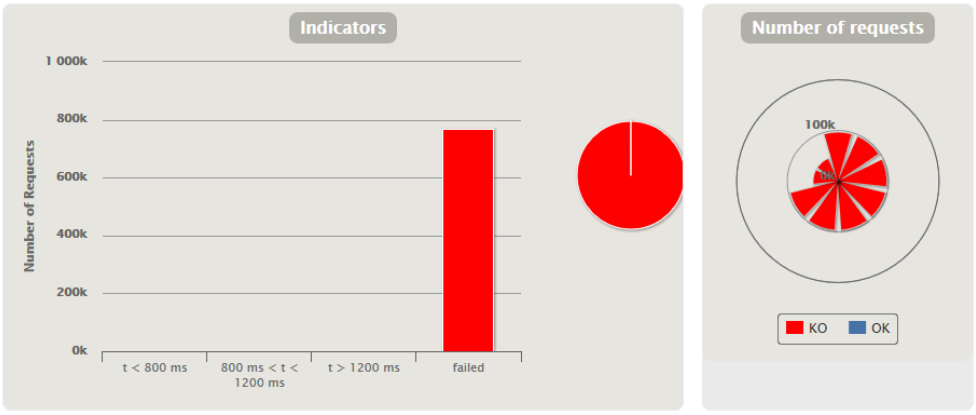


Figure 92 US-09 Stress test performance

  - Gatling Report



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	768000	273	767727	100%	2501.629	0	2340	4942	9740	60214	61858	3937	6502
home	96000	273	95727	100%	312.704	4358	5659	6154	60714	61366	61858	10611	15051
login	96000	0	96000	100%	312.704	0	180	2010	2407	2619	61785	1026	3821
findowners	96000	0	96000	100%	312.704	2	1073	3454	4574	4896	14117	1889	1559
listowners	96000	0	96000	100%	312.704	1	2075	3809	5839	8552	10825	2783	1598
ownerview	96000	0	96000	100%	312.704	2	2013	2993	5792	8936	9623	2619	1525
medicalhistory	96000	0	96000	100%	312.704	1	2718	3892	9755	13536	13802	3739	2700
medicalrecordshow	96000	0	96000	100%	312.704	19	4966	7378	12878	13251	13694	6016	3254
medicalr...negative	48000	0	48000	100%	156.352	1	2005	3000	8793	9455	10360	2811	2447
medicalr...positive	48000	0	48000	100%	156.352	0	2006	2967	8820	9506	10357	2808	2448

Figure 93 US-03 Stress test gatling results

- Load Test
  - System Performance

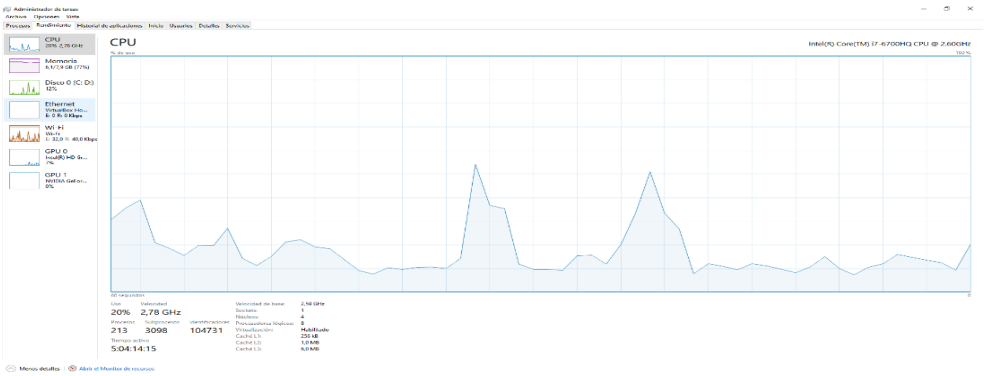
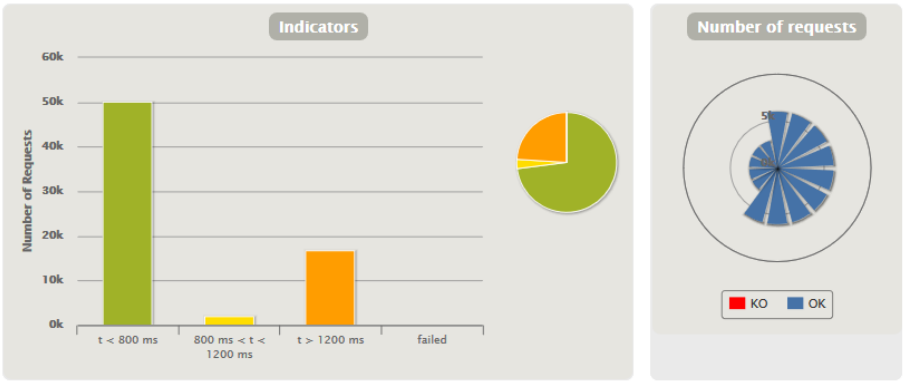


Figure 94 US-03 Load test performance

  - Gatling Report



ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	69000	69000	0	0%	258.427	0	6	1091	3754	5869	10408	753	1366
home	6000	6000	0	0%	22.472	1	3	4	7	11	89	4	3
login	6000	6000	0	0%	22.472	0	2	3	3787	6160	8536	306	1169
logged	6000	6000	0	0%	22.472	0	3	10	4302	6719	10295	694	1554
logged Redirect 1	6000	6000	0	0%	22.472	0	3	5	3099	3891	4009	410	968
findowners	6000	6000	0	0%	22.472	1	4	326	3809	6101	8692	692	1386
listowners	6000	6000	0	0%	22.472	6	733	2293	4504	6952	10329	1370	1626
ownerview	6000	6000	0	0%	22.472	2	322	1879	4264	6517	9710	1128	1535
medicalhistory	6000	6000	0	0%	22.472	1	354	2201	4526	6927	10408	1255	1658
medicalrecordshow	6000	6000	0	0%	22.472	2	374	2197	4441	6499	10009	1229	1597
medicalr...negative	3000	3000	0	0%	11.236	2	18	1498	2866	3962	6143	752	1070
medicalr...positive	3000	3000	0	0%	11.236	1	20	1495	2761	3787	7088	748	1052
error	3000	3000	0	0%	11.236	2	4	845	2757	3996	7405	590	1035
medicalr...dupdated	3000	3000	0	0%	11.236	1	5	790	2735	3775	6895	573	1009
medicalr...direct 1	3000	3000	0	0%	11.236	1	4	559	2430	3533	6268	488	905

Figure 95 US-03 Load test gatling results

US 00L

- Stress Test
    - System Performance

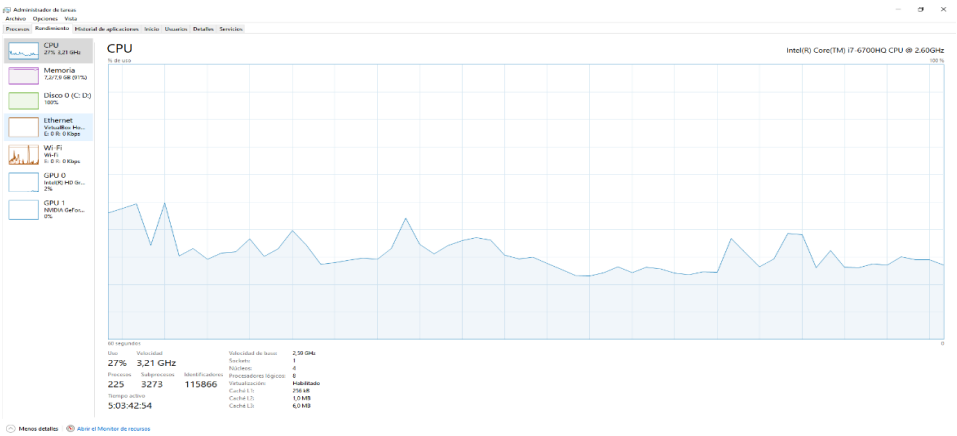
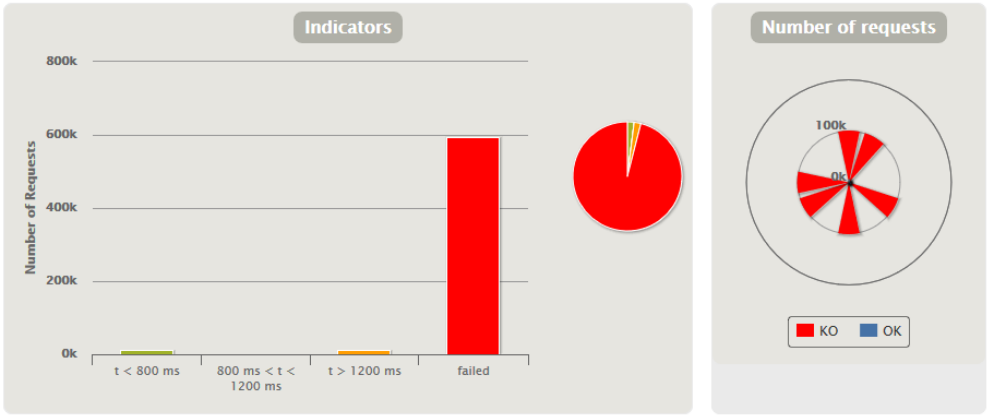


Figure 96 US-0L Stress test performance

    - Gatling Report



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	618995	25306	593689	96%	1551.366	0	2000	6556	16284	60038	80435	4801	8840
home	101200	5455	95745	95%	253.634	5934	7577	8006	35285	41716	42392	9828	7142
login	101200	4547	96653	96%	253.634	1	1388	1812	4961	5572	60006	1761	2231
logged	4547	2658	1889	42%	11.396	1	438	1354	60365	60951	62103	9758	21195
logged Redirect 1	2658	2655	3	0%	6.662	1	20	275	481	2059	60001	240	2037
findowners	101200	4879	96321	95%	253.634	0	1424	2391	5200	60279	72043	3002	8604
findowne...direct 1	4076	4076	0	0%	10.216	0	190	270	393	564	1416	194	121
listowners	101200	287	100913	100%	253.634	0	1541	3065	34407	60941	80435	6050	13700
listowne...direct 1	253	227	26	10%	0.634	1	1235	1416	60001	65968	71692	7583	18746
ownerview	101200	59	101141	100%	253.634	0	1559	3010	16370	30610	80215	3730	6821
medicalhistory	101200	202	100998	100%	253.634	0	2002	3906	18769	22386	60003	4524	7029
medicalh...direct 1	202	202	0	0%	0.506	1	31	35	42	45	49	24	14
ownervie...direct 1	59	59	0	0%	0.148	15	33	34	38	39	40	30	7

Figure 97 US-0L Stress test gatling results

- Load Test
  - System Performance

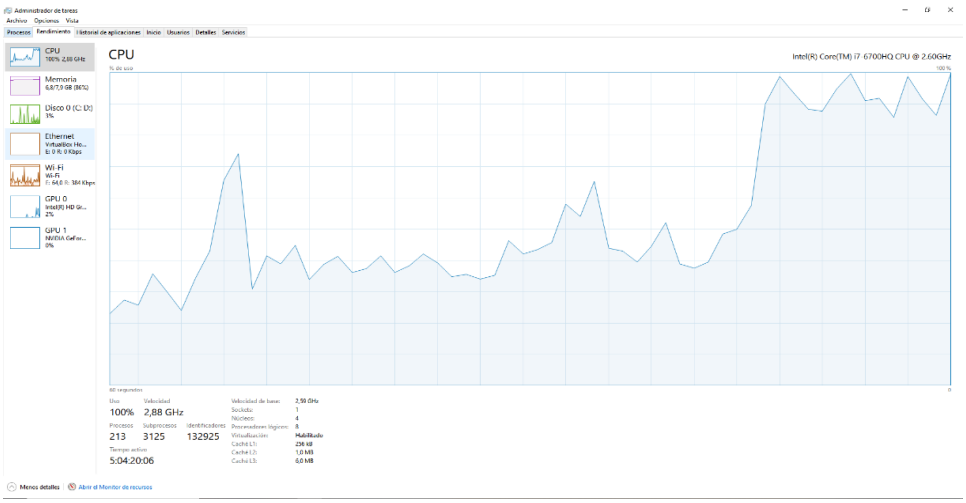
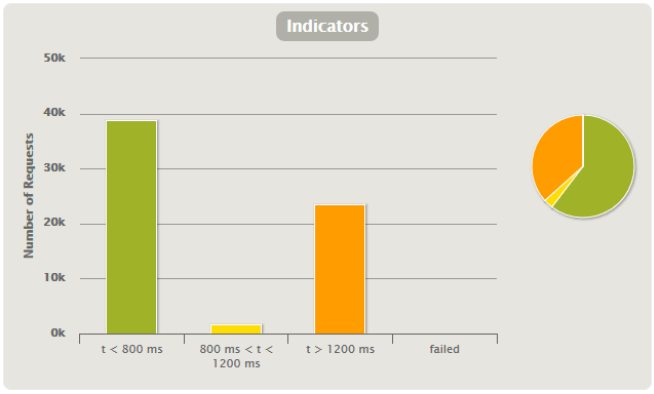
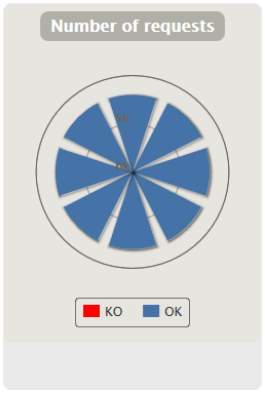


Figure 98 US-0L Load test performance

  - Gatling Report





ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 1000.0	KO
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	64000	64000	0	0%	266.667	0	110	3079	5202	6784	12676	1480	2001
home	8000	8000	0	0%	33.333	1	3	5	8	17	97	4	5
login	8000	8000	0	0%	33.333	0	2	1054	5036	5190	5240	949	1658
logged	8000	8000	0	0%	33.333	0	1023	3916	5746	7321	10559	1998	2184
logged Redirect 1	8000	8000	0	0%	33.333	0	7	3024	4868	5172	5241	1392	1794
findowners	8000	8000	0	0%	33.333	1	426	2982	4844	5177	5235	1446	1759
listowners	8000	8000	0	0%	33.333	5	1236	4007	5906	7523	12676	2160	2162
ownerview	8000	8000	0	0%	33.333	2	1081	3894	5974	7561	11741	2063	2215
medicalhistory	8000	8000	0	0%	33.333	2	327	3706	5809	7351	10836	1825	2228

Figure 99 US-0L Load test gatling results

US 00D

- Stress Test
  - System Performance

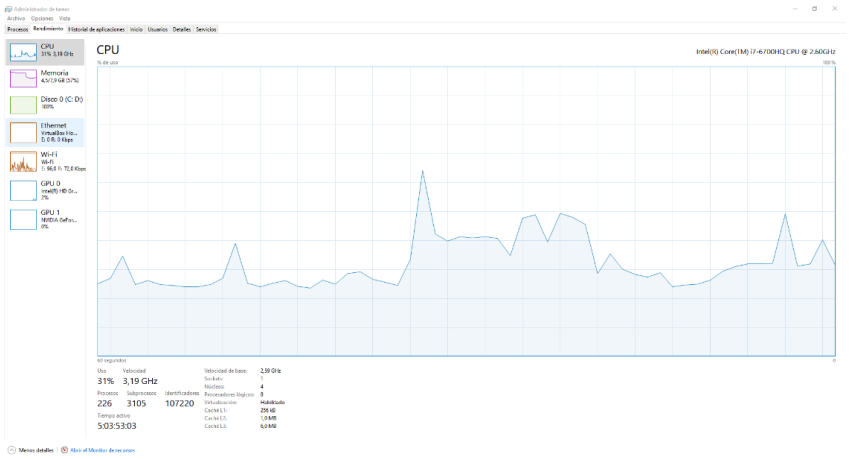
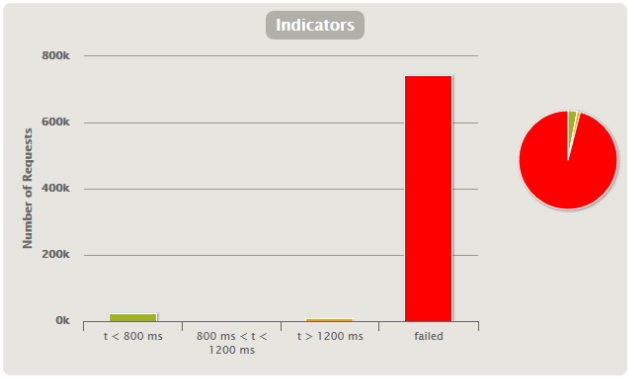
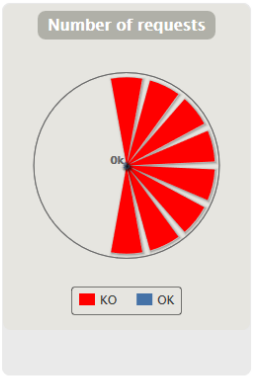


Figure 100 US-0D Stress test performance

  - Gatling Report





STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	775423	33214	742209	96%	2273.968	0	1736	3977	7872	60002	62411	3592	7084
home	95000	2368	92632	98%	278.592	5696	7033	7468	60642	61670	62411	11922	14733
login	95000	0	95000	100%	278.592	1	915	2001	4226	60007	61684	2756	9306
findowners	95000	396	94604	100%	278.592	1	1319	2295	3988	4261	25118	1842	1184
listowners	95000	485	94515	99%	278.592	0	1376	2371	4099	9934	23567	2000	1654
ownerview	95000	3379	91621	96%	278.592	0	1479	2374	4565	5084	26825	2044	1721
medicalhistory	95000	3460	91540	96%	278.592	0	1720	2435	4445	18815	23572	2324	2269
medicalrecordshow	95000	3577	91423	96%	278.592	0	1513	2072	4218	4825	23577	1958	1307
deleted	95000	4126	90874	96%	278.592	0	3900	5228	9103	9459	27267	4335	2609
ownervie...direct 1	3379	3379	0	0%	9.909	0	15	90	1775	7485	23571	423	1983
deleted Redirect 1	4126	4126	0	0%	12.1	0	40	160	1867	7513	23565	588	1926
findowne...direct 1	396	396	0	0%	1.161	0	2	15	425	1782	22988	158	1205
medicalh...direct 1	3460	3460	0	0%	10.147	0	35	1748	1894	7507	23580	879	2148
listowne...direct 1	485	485	0	0%	1.422	0	13	1755	7431	10113	23324	1079	2812
medicalr...direct 1	3577	3577	0	0%	10.49	0	33	168	7458	7817	23579	1558	3459

Figure 101 US-0D Stress test gatling results

- Load Test
    - System Performance

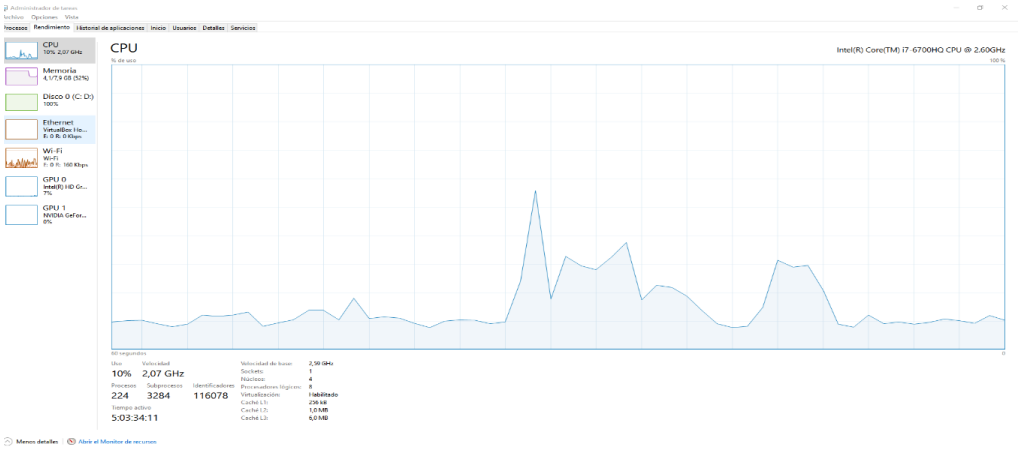


Figure 102 US-0D Load test performance

    - Gatling Report

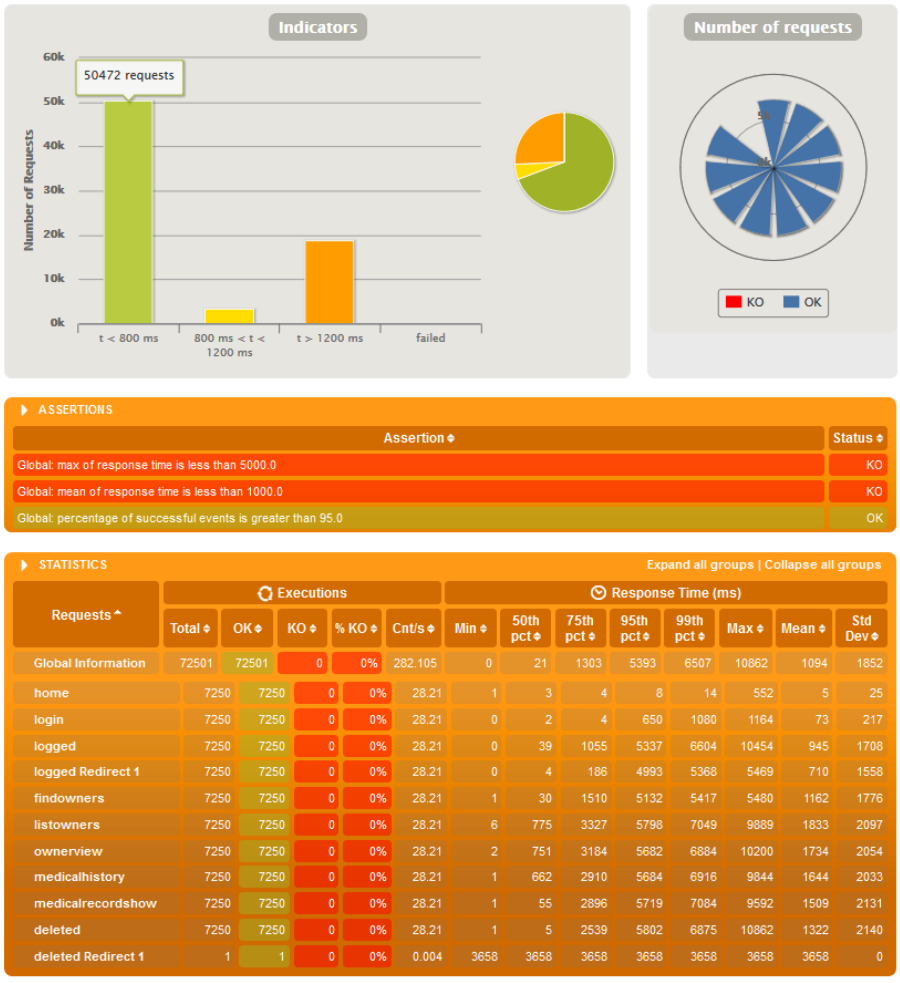


Figure 103 US-0D Load test gatling results

Conclusion

Results Summary

US and description	Street test	Load test
US 003: Medical record creation	100.000	6000
US 009: Vet changes a Pet's Medical Record	95.000	7250
US 00L: Listing for Medical Record	101.200	8000
US 00D: Deletion for Medical Record	96.000	6000

Looking at the result, we can conclude that the bottleneck is given by the RAM and the memory given to Garlick. While the CPU goes through some high and lows, the memory stays constantly high during all process, especially during Stress Test. Considering that I own a Laptop, It would be pretty difficult to change the RAM, so the solution would be using other PC.

Final Conclusion

US and description	Street test	Load test	Tester
US 001 Vet adds a new medicine	80000	2000	Manuel
US 002 Vet lists medicines	90000	2200 – 2500	Manuel
US 003: Medical record creation	100.000	6000	Iván
US 004 & US 005 & US 006: Listing for Medical Record	101.200	8000	Iván
US-007 & US-008 Vet prescribes medicines to a pet	80000	2000 – 2250	Manuel
US 009: Vet changes a Pet's Medical Record	95.000	7250	Iván
US 010 User Adopts a Pet	70000	1500	Álvaro
US 011 Homeless pet's management	85000	250	Yoana
US 012 Adoption procedure	85000	3000 – 3200	Manuel
US 013 Vet management	80000	1500	Yoana
US 014 Vet schedules new Intervention	65000	1500	Álvaro
US 015 Vet plans Intervention	50000	1500	Álvaro
US 016 Owners sees Interventions	70000	2000	Álvaro
US 017 Owners sees Vet's personal information	70000	2000	Álvaro
US 018: Vet sees pet's visits	60000	2000	Diāna
US 019 Trainer management	80000	3000	Yoana
US 020 Unregistered users can see trainers	100000	14000	Yoana
US 021: Training session organization	45000	1200	Diāna
US 022: Trainer plans rehabilitation sessions	50000	1300	Diāna
US 023: Owner sees rehabilitation sessions	70000	1900	Diāna
US 024: Owner sees trainer's personal information	50000	800	Diāna
US 025 Trainers can see pet's visits	80000	2000	Yoana
US 00D: Deletion for Medical Record	96.000	6000	Iván

After performing the testing for every single US and looking to the results, we can observe that the bottleneck is always either CPU, RAM or both. Also, we got fairly similar results, which makes sense because all of us have similar PC with similar powers.

Extra

As a disclaimer, because we made the document individually and then merged them together, the format and the pictures taken were very different, so there is some variation in the focus each one of us took.

Álvaro

For the purpose of the A+ task the group has decided to divide it and each one of use will work with his chosen US. In my case, the US I will be working with is US-15, vet plans new intervention.

The feature I will focus the extra task is the creation of the object, the intervention, with its positive and negative scenarios.

InterventionController.java | application.properties | Intervention.java

```
1
2  package org.springframework.samples.petclinic.model;
3
4  import java.time.LocalDate;
20
21 @Entity
22 @Data
23 @EqualsAndHashCode(callSuper = false)
24 @Table(name = "interventions")
25 public class Intervention extends BaseEntity {
26
27     @Column(name = "intervention_date")
28     @DateTimeFormat(pattern = "yyyy/MM/dd")
29     private LocalDate   interventionDate;
30
31     @Column(name = "intervention_time")
32     @NotNull
33     private Integer     interventionTime;
34
35     @Column(name = "intervention_description")
36     @NotEmpty
37     private String      interventionDescription;
38
39     @OneToOne(cascade = CascadeType.ALL)
40     private Vet         vet;
41
42     @ManyToOne
43     @JoinColumn(name = "pet_id")
44     private Pet         pet;
45 }
46
```

Figure 104 Intervention model

Intervention as a model is very simple and contains very few attributes so the main attention can be focused in the feeders and gatling.

With the previous knowledge about gatling, It is known that a scala file is necessary to run it, that will be saved in the simulation folder, whereas the files that we are going to use (csv format) will be contain in the resources file, so once gatling is running the scala script we have made, it will know where to find the information.

Scala file

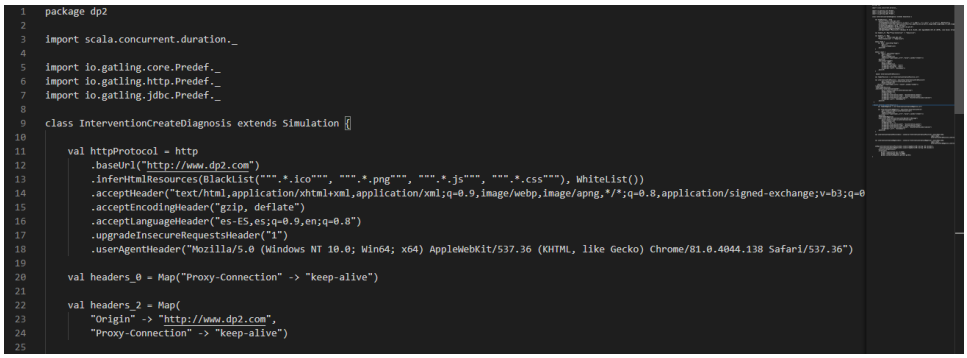
I have used Visual Studio Code to create the scala file and edit it.

To study this file, I have divided it in 4 parts: definition of values and headers used, objects, scenarios (positive and negatives), setUp.

Definition of values and headers

Here it is very simple, first, we need the http protocol, which it is already provided by Gatling, so there is little work to do, just revise the blacklist.

The other headers are also created automatically.



```
package dp2

import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class InterventionCreateDiagnosis extends Simulation {

    val httpProtocol = http
        .baseUrl("http://www.dp2.com")
        .inferHtmlResources(BlackList("""*.ico""", """*.png""", """*.js""", """*.css"""), WhiteList())
        .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0
        .acceptEncodingHeader("gzip, deflate")
        .acceptLanguageHeader("es-ES,es;q=0.9,en;q=0.8")
        .upgradeInsecureRequestsHeader("1")
        .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36")

    val headers_0 = Map("Proxy-Connection" -> "keep-alive")

    val headers_2 = Map(
        "Origin" -> "http://www.dp2.com",
        "Proxy-Connection" -> "keep-alive")

```

Figure 105 Scala headers

Objects

As in performance testing, we have to define how we get to the application, the home page and login with the correct user for the operation we want to perform.

Scenarios

The scenarios consist on a feeder that reads the csv file and a method that calls a url to perform a POST operation in the application, the reads the csv file and inject the values in the attributes we have defined in the model of Intervention.

Although intervention also has a vet as a attribute for the purpose of having less complexity, and also been a non-necessary attribute, I have avoided working with it.

Positive

In the positive scenario the csv file contains correct data and when the file its runned there are no errors.

```
object InterventionFormPositive {

  val feederPositive = csv("InterventionCreationPositive.csv")

  val interventionFormPositive = exec(http("InterventionFormPositive")
          .get("/owners/*/pets/1/interventions/new")
          .headers(headers_0)
    .check(css("input[name=_csrf]","value").saveAs("stoken"))
  ).pause(34)
  .feed(feederPositive)
  .exec(http("InterventionCreated")
          .post("/owners/*/pets/1/interventions/new")
          .headers(headers_0)
          .formParam("id", "")
          .formParam("intervention_date", "${intervention_date}")
          .formParam("intervention_time", "${intervention_time}")
          .formParam("intervention_description", "${intervention_description}")
          .formParam("_csrf", "${stoken}"))
      .pause(5)
}
```

Figure 106 Changes made to Scala (Positive)

Negative

In the negative scenario the csv file contains incorrect data and when the file its runned there are errors.

```
object InterventionFormNegative {
      var feederNegative = csv("InterventionCreationNegative.csv")

      var interventionFormNegative = exec(http("InterventionForm")
          .get("/owners/*/pets/1/interventions/new")
          .headers(headers_0)
          .check(css("input[name=_csrf]","value").saveAs("stoken")))
      .pause(26)
      .feed(feederNegative)
      .exec(http("InterventionCreationFormWithErrorMessage")
          .post("/owners/*/pets/1/interventions/new")
          .headers(headers_0)
          .formParam("id", "")
          .formParam("intervention_date", "${intervention_date}")
          .formParam("intervention_time", "${intervention_time}")
          .formParam("intervention_description", "${intervention_description}")
          .formParam("_csrf", "${stoken}"))
      .pause(10)
   }
```

Figure 107 Figure 106 Changes made to Scala (Negative)

SetUp

Inside the setUp part we have two things.

The first one is the code that executes the scenarios when the file is ran. It calls the object we have defined in the file in the order we need, in this case: Home, Login and InterventionFormPositive/Negative.

The second part is the configuration of the simulation scenario. This is done by performing changes in the assertions part. First, we limit the time in which we want the operation to be done and we define the percentage of expected requests.

```
    val interventionCreationPositiveScn = scenario("InterventionCreationPositive").exec(Home.home,
                                                                                        Login.login,
                                                                                        InterventionFormPositive.interventionFormPositive)

    val interventionCreationNegativeScn = scenario("InterventionCreationNegative").exec(Home.home,
                                                                                        Login.login,
                                                                                        InterventionFormNegative.interventionFormNegative)

    setUp(interventionCreationPositiveScn.inject(rampUsers(40) during (50 seconds)),
          interventionCreationNegativeScn.inject(rampUsers(40) during (50 seconds)))
        .protocols(httpProtocol)
        .assertions(
            global.responseTime.max.lt(5000),
            global.responseTime.mean.lt(1000),
            global.successfulRequests.percent.gt(95))
}
```

Figure 108 Scala Set Up

CSV files

The csv files have been done using www.mockaroo.com.

I have simply defined the attributed I want to appeared in the csv file, define its attributes, its values and the rows I want.

Positive scenario:

Just the correct data.

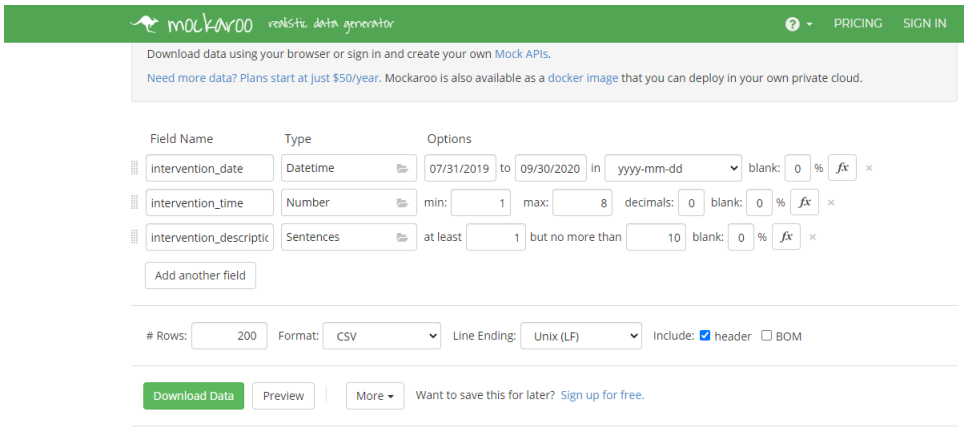


Figure 109 Inserting values with Mockaroo (Positive)

Negative scenario:

Here I just ensure that there will be blank data, that will create an error once you try to create the intervention.

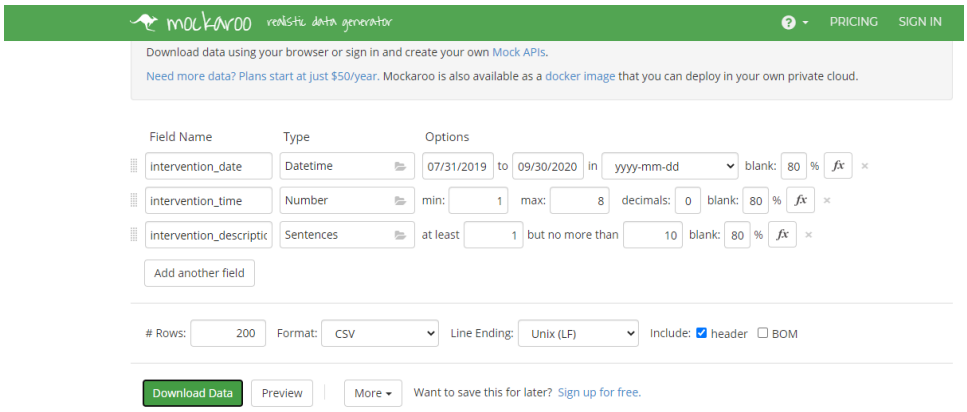


Figure 110 Inserting values with Mockaroo (Negative)

Result

Once the file is running the console start working on the requests.

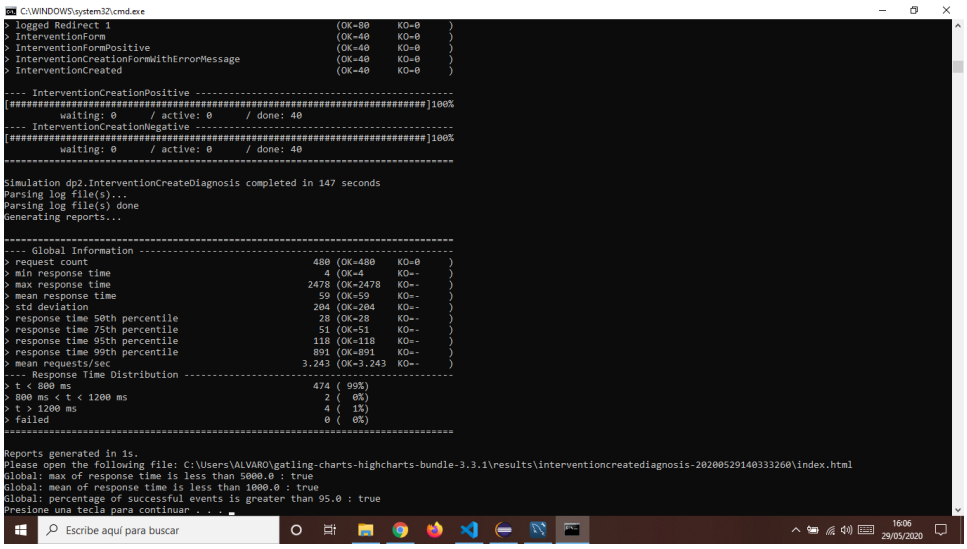


Figure 111 Gatling is executing correctly

The result of Gatlin is saved in a result file. Here we have the statistics of how the process went.

Figure 112 Global Analysis

As we can see here, the assertions defined have status OK.

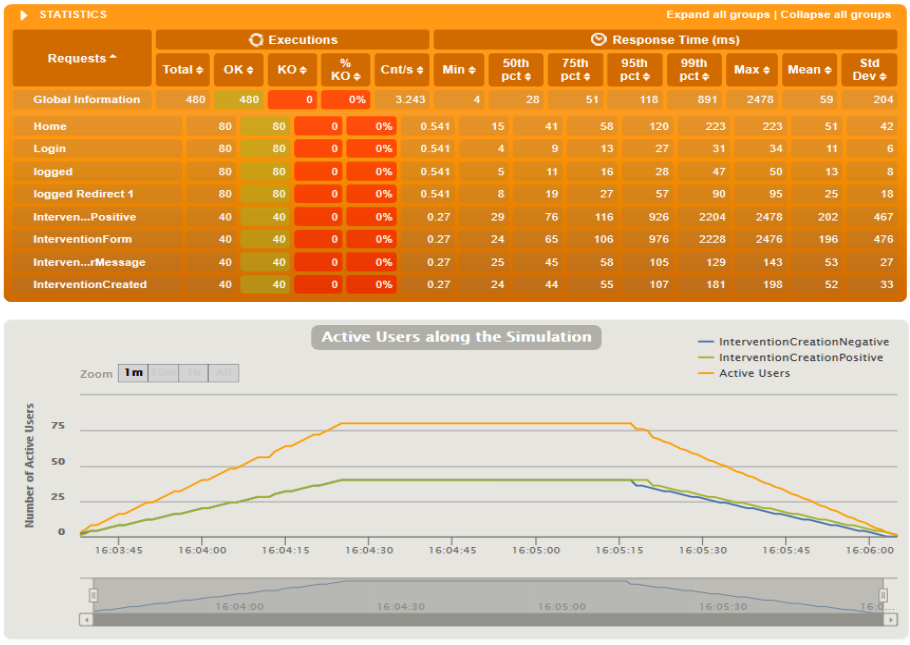


Figure 113 Results

In the result, we observe that we have 40 Interventions Created and 40 messages (this are error when creating an intervention) thus, with this data we know that the csv files work.

Yoana

This report is dedicated to the extra task of this deliverable, the A+. The theme is Gatling feeders. Before doing the task itself I wanted to make a little research. I used the following links for that:

- https://gatling.io/docs/current/session/feeder
- https://gatling.io/docs/current/advanced_tutorial

I also used an additional resource for making the CSV files. The link for it is the following:

- https://www.mockaroo.com/

So, the first step was actually making the CSV files. You can check the configuration I used for that in Mockaroo in the next screenshot:

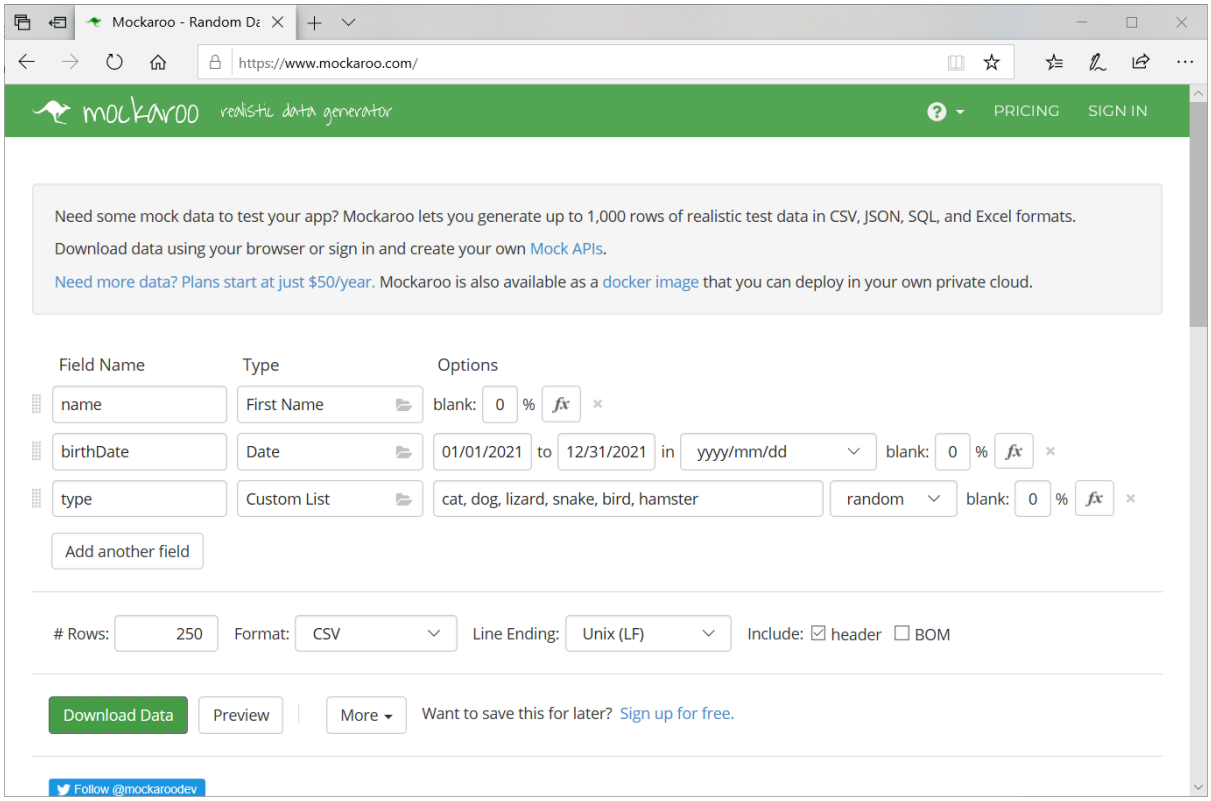


Figure 114 Inserting values in Mockaroo

The user story for which I made this is US – 011 Homeless Pet Management in which I had the positive and negative scenario of creating a new pet. For that I needed a different name each time I inserted a new pet. And also, I used random values for the other fields to make it more interesting. The configuration in the previous screenshot is actually only for the negative scenario because the birth dates are in future (impossible to insert a pet with that characteristics). The procedure for the positive scenario is similar anyways.

The second step was actually modifying the Scala file of the scenarios. I needed to add the things in red in the next screenshots:

```
object HomelessPetFormPositive {

  val feederPositive = csv("HomelessPetManagementPositive.csv")

  val homelessPetFormPositive = exec(http("HomelessPetFormPositive")
    .get("/homeless-pets/new")
    .headers(headers_0)
    .check(css("input[name=_csrf]","value").saveAs("stoken"))
  ).pause(34)
  .feed(feederPositive)
  .exec(http("HomelessPetCreated")
    .post("/homeless-pets/new")
    .headers(headers_3)
    .formParam("id", "")
    .formParam("name", "${name}")
    .formParam("birthDate", "${birthDate}")
    .formParam("type", "${type}")
    .formParam("_csrf", "${stoken}"))
  .pause(5)
}
```

Figure 115 Changes in scala (Positive)

```
object HomelessPetFormNegative {

  val feederNegative = csv("HomelessPetManagementNegative.csv")

  val homelessPetFormNegative = exec(http("HomelessPetFormNegative")
    .get("/homeless-pets/new")
    .headers(headers_0)
    .check(css("input[name=_csrf]","value").saveAs("stoken"))
  ).pause(34)
  .feed(feederNegative)
  .exec(http("HomelessPetFormWithErrorMessage")
    .post("/homeless-pets/new")
    .headers(headers_3)
    .formParam("id", "")
    .formParam("name", "${name}")
    .formParam("birthDate", "${birthDate}")
    .formParam("type", "${type}")
    .formParam("_csrf", "${stoken}"))
  .pause(28)
}
```

Figure 116 Changes in scala (Negative)

To edit the Scala file, I used this resource: https://scastie.scala-lang.org/

The third step was to actually start the database and launch the application and after that I executed the script saved before.

These are the pets before executing the script:

```
mysql> select * from pets;
+----+----------+------------+----------+---------+
| id | name     | birth_date | owner_id | type_id |
+----+----------+------------+----------+---------+
|  1 | Leo      | 2010-09-07 |        1 |       1 |
|  2 | Basil    | 2012-08-06 |        2 |       6 |
|  3 | Rosy     | 2011-04-17 |        3 |       2 |
|  4 | Jewel    | 2010-03-07 |        3 |       2 |
|  5 | Iggy     | 2010-11-30 |        4 |       3 |
|  6 | George   | 2010-01-20 |        5 |       4 |
|  7 | Samantha | 2012-09-04 |        6 |       1 |
|  8 | Max      | 2012-09-04 |        6 |       1 |
|  9 | Lucky    | 2011-08-06 |        7 |       5 |
| 10 | Mulligan | 2007-02-24 |        8 |       2 |
| 11 | Freddy   | 2010-03-09 |        9 |       5 |
| 12 | Lucky    | 2010-06-24 |       10 |       2 |
| 13 | Sly      | 2012-06-08 |       10 |       1 |
| 14 | Tucker   | 2018-06-08 |     NULL |       2 |
| 15 | Lekay    | 2016-04-04 |     NULL |       1 |
| 16 | Miss     | 2015-03-17 |     NULL |       1 |
+----+----------+------------+----------+---------+
16 rows in set (0.00 sec)
```

Figure 117 Database before executing the script

After executing the script, the following screenshot shows part of the query result I got from the database:

```
mysql> select * from pets;
+----+-------------+------------+----------+---------+
| id | name        | birth_date | owner_id | type_id |
+----+-------------+------------+----------+---------+
|  1 | Leo         | 2010-09-07 |        1 |       1 |
|  2 | Basil       | 2012-08-06 |        2 |       6 |
|  3 | Rosy        | 2011-04-17 |        3 |       2 |
|  4 | Jewel       | 2010-03-07 |        3 |       2 |
|  5 | Iggy        | 2010-11-30 |        4 |       3 |
|  6 | George      | 2010-01-20 |        5 |       4 |
|  7 | Samantha    | 2012-09-04 |        6 |       1 |
|  8 | Max         | 2012-09-04 |        6 |       1 |
|  9 | Lucky       | 2011-08-06 |        7 |       5 |
| 10 | Mulligan    | 2007-02-24 |        8 |       2 |
| 11 | Freddy      | 2010-03-09 |        9 |       5 |
| 12 | Lucky       | 2010-06-24 |       10 |       2 |
| 13 | Sly         | 2012-06-08 |       10 |       1 |
| 14 | Tucker      | 2018-06-08 |     NULL |       2 |
| 15 | Lekay       | 2016-04-04 |     NULL |       1 |
| 16 | Miss        | 2015-03-17 |     NULL |       1 |
| 17 | Annabal     | 2018-04-13 |     NULL |       3 |
| 18 | Roxine      | 2019-07-24 |     NULL |       4 |
| 19 | Marrissa    | 2019-07-28 |     NULL |       4 |
| 20 | Goldarina   | 2019-07-06 |     NULL |       4 |
| 21 | Kip         | 2019-03-12 |     NULL |       6 |
| 22 | Grier       | 2019-11-25 |     NULL |       3 |
| 23 | Lianne      | 2019-03-24 |     NULL |       6 |
| 24 | Estel       | 2019-10-21 |     NULL |       3 |
| 25 | Walker      | 2019-10-01 |     NULL |       6 |
| 26 | Les         | 2019-05-23 |     NULL |       2 |
| 27 | Keri        | 2019-05-16 |     NULL |       2 |
| 28 | Ransom      | 2018-08-10 |     NULL |       6 |
| 29 | Meris       | 2018-11-30 |     NULL |       5 |
| 30 | Julietta    | 2019-07-03 |     NULL |       3 |
| 31 | Henryetta   | 2018-09-08 |     NULL |       6 |
| 32 | Libbi       | 2018-04-18 |     NULL |       4 |
| 33 | Carlin      | 2018-12-03 |     NULL |       6 |
| 34 | Bradley     | 2018-09-19 |     NULL |       3 |
| 35 | Edgar       | 2020-03-11 |     NULL |       1 |
| 36 | Carney      | 2019-04-17 |     NULL |       2 |
```

Figure 118 Database after executing the script

And we can also see them in the actual page:

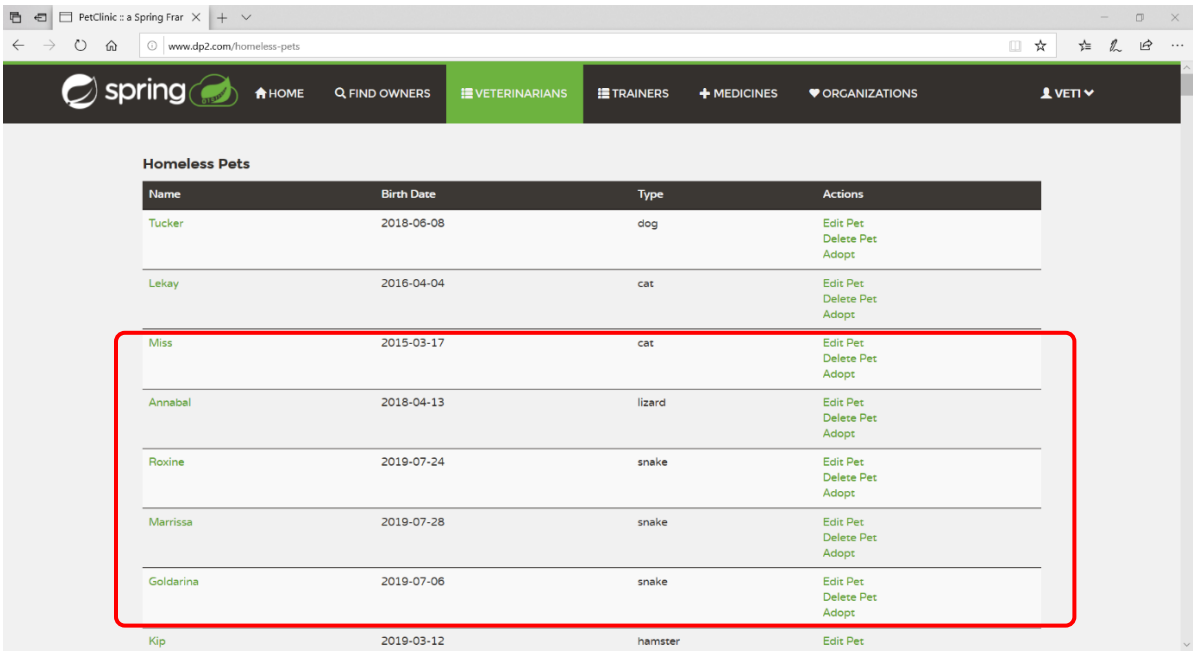


Figure 119 Values updated in the page

We can also check the performance results while using feeders. You can see the CPU overall performance during the execution of the script and some data from the Gatling reports on the next page.

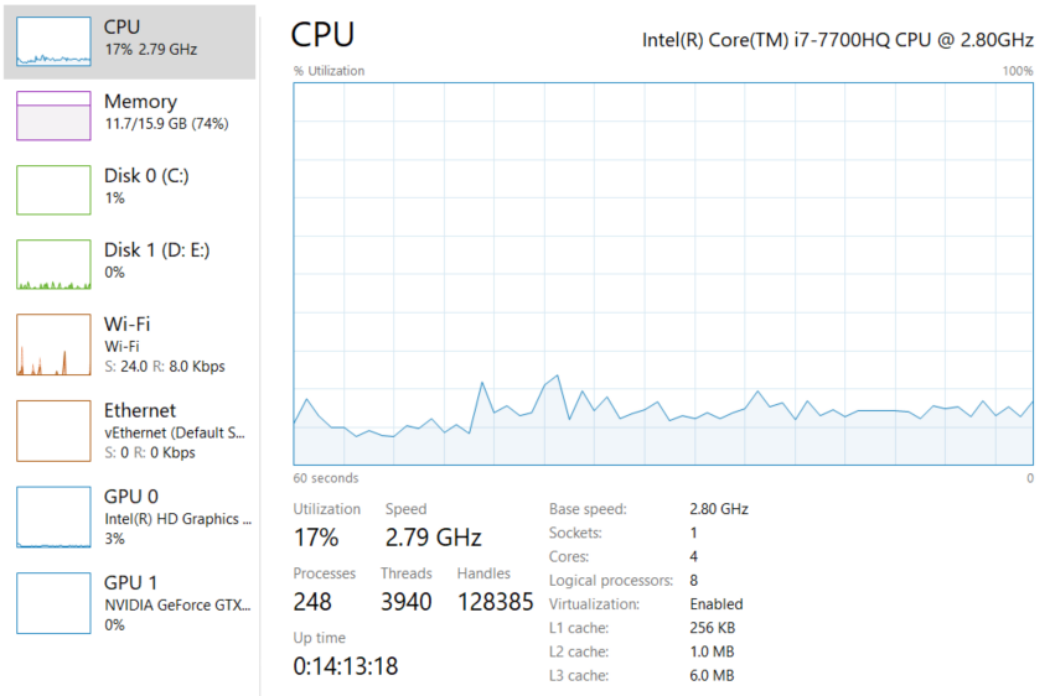


Figure 120 Analysis

> Global Information

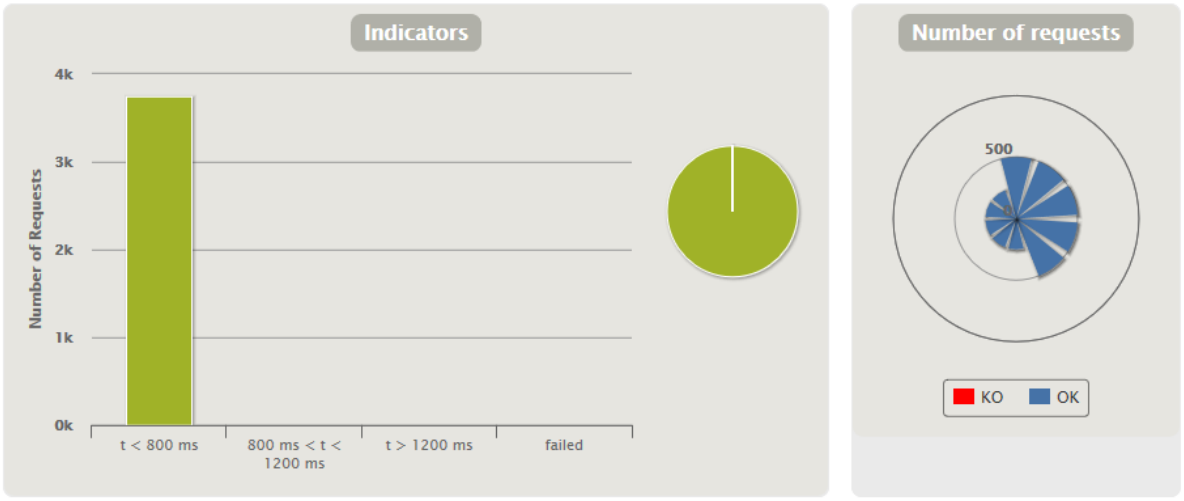


Figure 121 Global analysis

▸ ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	OK
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

▸ STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	3749	3749	0	0%	15.686	0	12	23	47	135	1314	20	54
Home	500	500	0	0%	2.092	2	5	7	16	670	1314	19	114
Login	500	500	0	0%	2.092	0	2	2	4	20	28	2	3
Logged	500	500	0	0%	2.092	4	8	11	18	48	287	11	23
Logged Redirect 1	500	500	0	0%	2.092	1	3	4	8	22	25	4	3
ListHomelessPets	500	500	0	0%	2.092	11	22	33	62	452	464	35	61
Homeless...Positive	250	250	0	0%	1.046	9	16	22	32	387	409	26	54
Homeless...Negative	250	250	0	0%	1.046	9	15	22	34	393	401	26	55
Homeless...rMessage	250	250	0	0%	1.046	16	25	36	48	53	74	29	10
HomelessPetCreated	250	250	0	0%	1.046	20	30	44	57	132	135	37	18
Homeless...direct 1	249	249	0	0%	1.042	12	33	44	58	70	82	34	13

Figure 122 Details

Diāna

For the Gatling extra task or Gatling Feeders I choose US-21 (Training session organization), which means trainer adding new rehabilitations, since I it is a suitable user story for generating test cases, since there is an input.

As for the scala file, I took a previously added performance test diagnosis scala file for US-21 and made some changes in order to make Gatling feeder applicable.

That included changes:

1) Declaring new variables so that when the scala file's script is executed, then the data will be taken from the 2 csv files defined below:

   val feederPositive = csv("RehabManagementPositive4.csv")
   val feederNegative = csv("RehabManagementNegative4.csv")

2) Adding the actual 'call' for the csv files, so that when the script is running, then the data inside the csv will be inserted into the new rehabs that are going to be added.

   .feed(feederPositive)
   .feed(feederNegative)

3) Changing the parameters, so that for each rehab different data is inserted instead of a previously defined constants

   In the screenshots below there is a scala file with the added Gatling feeder extensions:

```
object AddedSuccessfulNewRehab {

val feederPositive = csv("RehabManagementPositive4.csv")

  val addedSuccessfulNewRehab2 = exec(http("AddedSuccessfulNewRehab")
      .get("/owners/3/pets/3/rehab/new")
      .headers(headers_0)
      .check(css("input[name=_csrf]","value").saveAs("stoken"))
    ).pause(34)
.feed(feederPositive)
.exec(http("AddedSuccessfulrehab")
      .post("/owners/3/pets/3/rehab/new")
      .headers(headers_3)
      .formParam("petId", "3")
      .formParam("id", "")
      .formParam("date", "${date}")
      .formParam("time", "${time}")
      .formParam("description", "${description}")
      .formParam("_csrf", "${stoken}"))
    .pause(37)
```

Figure 123 Adding rehab positive case

```
object UnsuccessfulAddingOfRehab {

val feederNegative = csv("RehabManagementNegative4.csv")

 val addedUnsuccessfulRehab = exec(http("UnsuccessfulAddingOfRehab")
      .get("/owners/3/pets/3/rehab/new")
      .headers(headers_0)
      .check(css("input[name=_csrf]","value").saveAs("stoken"))
    ).pause(34)
.feed(feederNegative)
.exec(http("UnsuccessfulAddingOfRehab2")
      .post("/owners/3/pets/3/rehab/new")
      .headers(headers_3)
      .formParam("petId", "3")
      .formParam("id", "")
      .formParam("date", "${date}")
      .formParam("time", "${time}")
      .formParam("description", "${description}")
      .formParam("_csrf", "${stoken}"))
    .pause(27)
```

Figure 124 Adding rehab negative case

For generating csv files, the website https://www.mockaroo.com/ was used.

Before running the script, I checked the database state, to see how many rehabs are saved as for now. We can see that there are 2 rehabs.

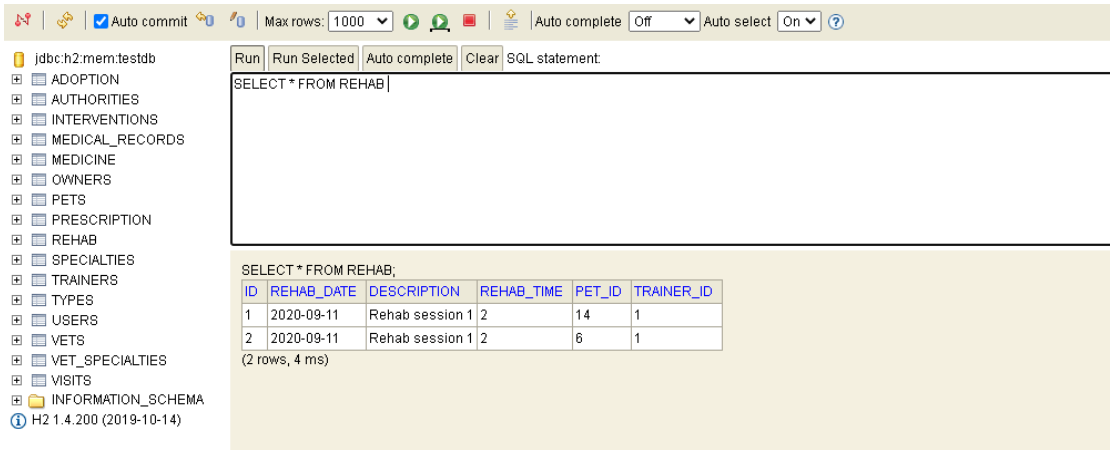


Figure 125 Rehabs before adding Gatling feeder

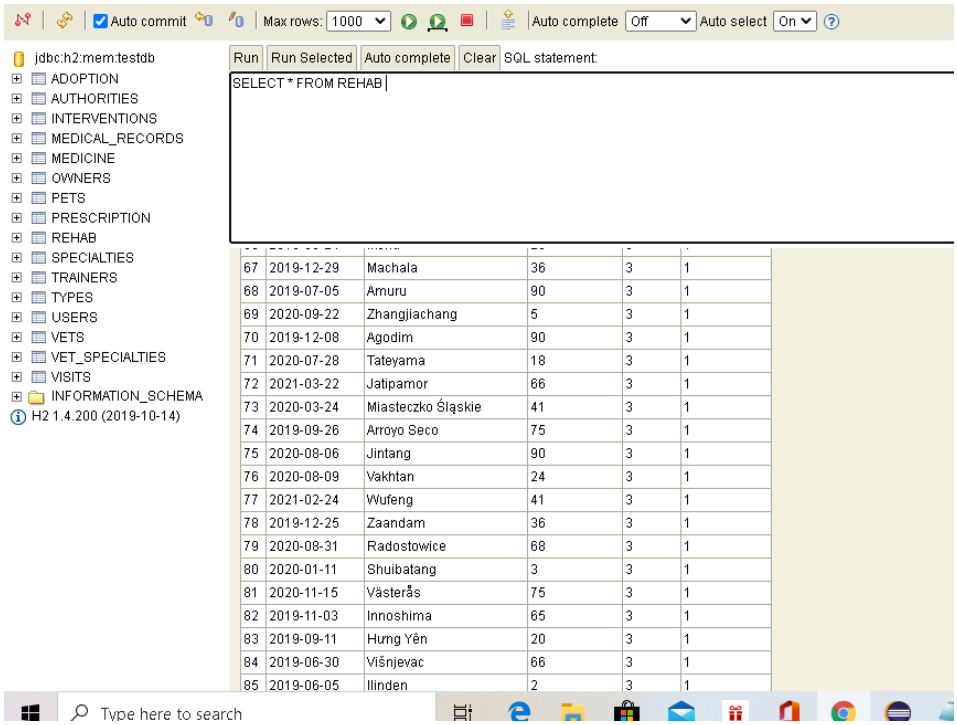


Figure 126 Rehabs after adding Gatling feeder 2

After executing the script, I checked the database again to see if there are any changes regarding to rehabs, as there should be added rehabs. We can see that many rehabs have been added:

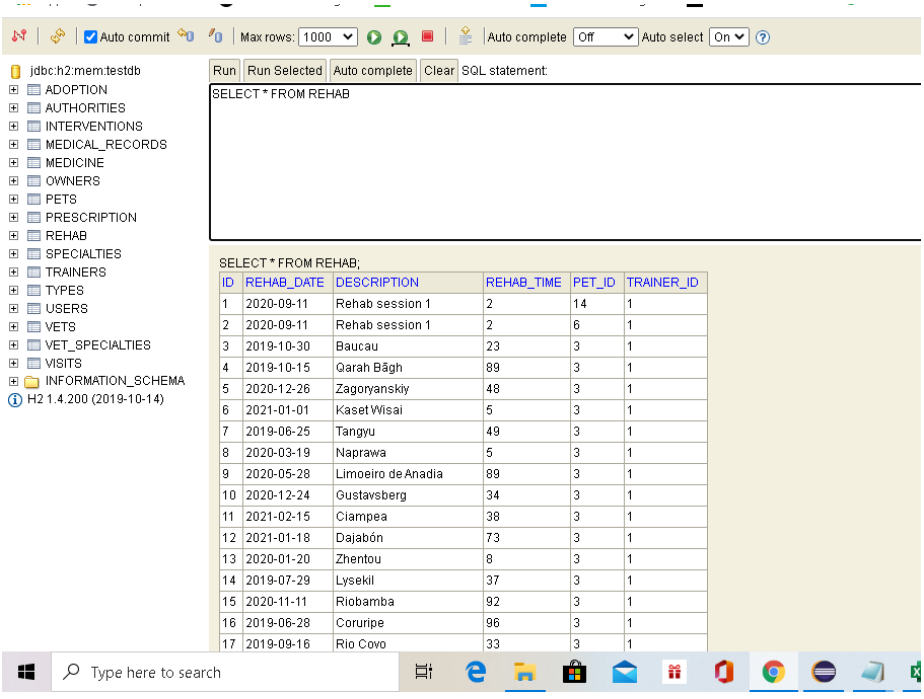


Figure 127 Rehabs after adding Gatling feeder 1

The same situation we can see when we log into our petclinic website,that pet with an ID 3 has a lot of rehabs added:

Figure 128 Petclinic website after adding Gatling feeder

Manuel

The object of this report is to describe the way we can test a massive number of users posting real data during a stress or load test.

The first step is finding an external tool to generate csv files containing the data for the simulation. In this case, I used mockaroo (https://www.mockaroo.com/) which is set up like it follows:

For positive cases generator:

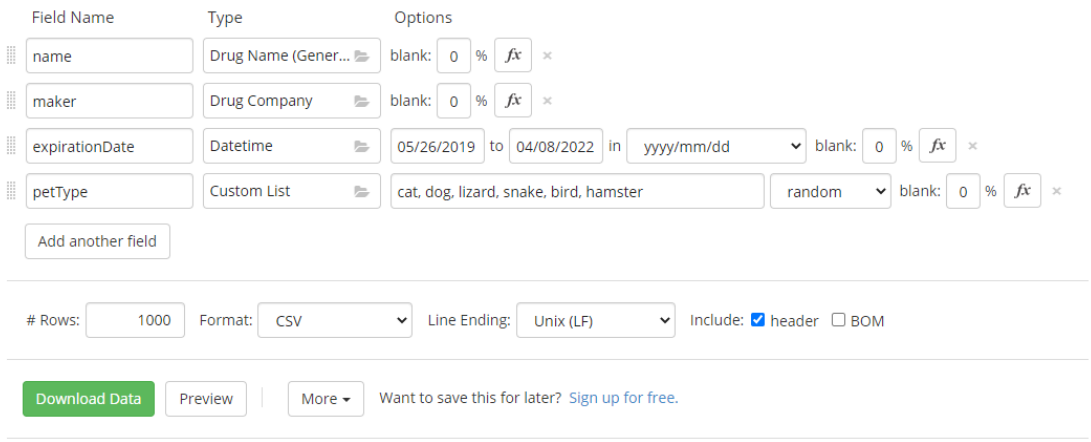


Figure 129 Inserting data in Moockaroo (Positive)

For negative cases generator:

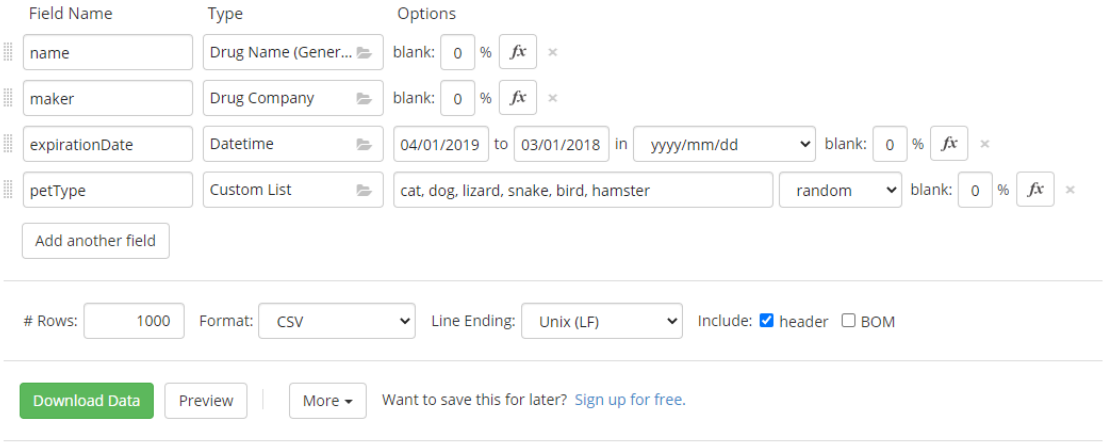


Figure 130 Inserting data in Moockaroo (Negative)

Once we have configured our fields with the necessary restrictions, we download the data.

Some entries of the generated file for the positive cases are…

```
DENSIFLORUM FLOWERING TOP",Hevert Pharmaceuticals LLC,2020/12/07,bird
benzocaine and menthol,Combe Incorporated,2021/08/27,lizard
Buspirone Hydrochloride,Dr. Reddy's Laboratories Ltd,2021/02/07,dog
Sodium chloride,United Exchange Corp,2021/06/07,lizard
Pumpkin,"Nelco Laboratories, Inc.",2020/11/07,cat
sodium fluoride,3M ESPE Dental Products,2020/12/19,hamster
Phentolamine Mesylate,Bedford Laboratories,2020/10/30,hamster
"Coniferyl alcohol,",Apotheca Company,2021/06/07,snake
"Titanium Dioxide, Zinc Oxide, and Octinoxate",Ventura International LTD.,2020/12/09,lizard
povidone iodine,CareFusion 213 LLC,2021/12/02,dog
not applicable,BioActive Nutritional Inc.,2019/08/02,dog
Allergenic Extracts Alum Precipitated,"ALK-Abello, Inc.",2021/01/23,dog
Epinastine Hydrochloride,"Breckenridge Pharmaceutical, Inc.",2021/05/30,bird
Meprobamate,Heritage Pharmaceuticals Inc.,2019/09/08,lizard
Menthol,U.S.A. AH LLC,2021/07/08,bird
ATORVASTATIN CALCIUM,Unit Dose Services,2020/09/06,cat
Captopril,Bryant Ranch Prepack,2021/04/17,cat
```

Figure 131 Examples (Positive)

… and the negative cases are:

```
name,maker,expirationDate,type
Sodium Fluoride,PureTek Corporation,2018/08/23,hamster
TITANIUM DIOXIDE,"Melaleuca, Inc.",2017/11/20,snake
lactulose,Cumberland Pharmaceuticals Inc.,2016/10/05,lizard
divalproex sodium,"Physicians Total Care, Inc.",2019/06/12,snake
DIMETHICONE,"Lansinoh Laboratories, Inc.",2019/03/24,snake
Isosorbide Dinitrate,Bryant Ranch Prepack,2017/05/31,snake
HYDROCORTISONE BUTYRATE,Onset Dermatologics LLC,2017/05/27,lizard
Acetaminophen,"Physicians Total Care, Inc.",2018/09/02,hamster
AMLODIPINE BESYLATE,KAISER FOUNDATION HOSPITALS,2019/10/22,bird
Sodium Chloride,REMEDYREPACK INC.,2017/09/10,hamster
Diphenhydramine Hydrochloride,"SIMPLY RIGHT (Sam's West, Inc.)",2019/06/24,snake
Buspirone hydrochloride,State of Florida DOH Central Pharmacy,2019/04/03,lizard
Ethyl Alcohol,"Soaptronic, LLC",2019/05/10,dog
Latanoprost,Bausch & Lomb Incorporated,2020/01/04,dog
```

Figure 132 Examples (Negative)

The sole difference between the documents is the date, which is in the future for the positive cases and in the past for the negative ones.

To use this data, we need to make some changes on the scala script we are going to use the data. Those changes are marked in red:

```
object MedicineCreationFormPositive {
    var feederPositive = csv("MedicineCreationPositive.csv")

    var medicineCreationFormPositive = exec(http("MedicineForm")
        .get("/medicine/create")
        .headers(headers_0)
        .check(css("input[name=_csrf]","value").saveAs("stoken")))
    .pause(26)
    .feed(feederPositive)
    .exec(http("MedicineCreated")
        .post("/medicine/create")
        .headers(headers_2)
        .formParam("name", "${name}")
        .formParam("expirationDate", "${expirationDate}")
        .formParam("maker", "${maker}")
        .formParam("petType", "${petType}")
        .formParam("_csrf", "${stoken}"))
    .pause(10)
}
```

Figure 133 Changes made to scala

The first marked line is for loading the data file, the second one is for reading the values and the last 4 are for inject the read data.

Once we have made those changes, we execute the scripts and we can notice how the database is populated with all the new data:

id	name	expiration_date	maker	type_id
1	Cat medicine	2023-05-22	Maker	1
2	Dog medicine	2023-05-22	Maker	2
3	benzocaine and menthol	2021-08-26	Combe Incorporated	3
4	Buspirone Hydrochloride	2021-02-06	Dr. Reddy's Laboratories Ltd	2
5	Sodium chloride	2021-06-06	United Exchange Corp	3
6	Pumpkin	2020-11-06	Nelco Laboratories, Inc.	1
7	sodium fluoride	2020-12-18	3M ESPE Dental Products	6
8	Phentolamine Mesylate	2020-10-29	Bedford Laboratories	6
9	Coniferyl alcohol,	2021-06-06	Apotheca Company	4
10	Titanium Dioxide, Zinc Oxide, and Octinoxate	2020-12-08	Ventura International LTD.	3
11	povidone iodine	2021-12-01	CareFusion 213 LLC	2
12	Allergenic Extracts Alum Precipitated	2021-01-22	ALK-Abello, Inc.	2
13	Epinastine Hydrochloride	2021-05-29	Breckenridge Pharmaceutic...	5
14	Menthol	2021-07-07	U.S.A. AH LLC	5
15	ATORVASTATIN CALCIUM	2020-09-05	Unit Dose Services	1
16	Captopril	2021-04-16	Bryant Ranch Prepack	1
17	Chloroprocaine Hydrochloride	2022-01-07	General Injectables & Vacci...	3
18	leflunomide	2022-03-30	Bryant Ranch Prepack	1
19	rifampin	2020-09-26	Lupin Pharmaceuticals, Inc.	2
20	MINOXIDIL	2020-11-18	Pure Source Inc.	4
21	Triclocarban	2021-08-30	VVF Illinois Services LLC	4
22	fexofenadine hydrochloride	2021-04-02	PD-Rx Pharmaceuticals, Inc.	6
23	Aluminum Sesquichlorohydrate	2021-12-19	Ventura Corporation LTD.	3
24	Paroxetine hydrochloride	2020-10-04	Cardinal Health	6
25	Labetalol hydrochloride	2022-04-19	Bryant Ranch Prepack	2
26	Promethazine Hydrochloride	2021-05-30	A-S Medication Solutions LLC	2
27	Pyrithione Zinc	2020-08-14	Wakefern Food Corp	6
28	Topiramate	2021-04-21	REMEDYREPACK INC.	6
29	GLYCERIN	2022-03-03	Happy Sonic Global Co., Ltd.	6
30	Valproic Acid	2021-05-20	Sun Pharmaceutical Industri...	3
31	LIDOCAINE HYDROCHLORIDE, MENTHOL	2021-07-27	Pharmaceutics Corporation	4

Figure 134 Updated database

Iván

For the A+ we decided to go with Gatling feeders. The first step is making the CSV files, which were made using Mookaroo (link: https://www.mockaroo.com/).

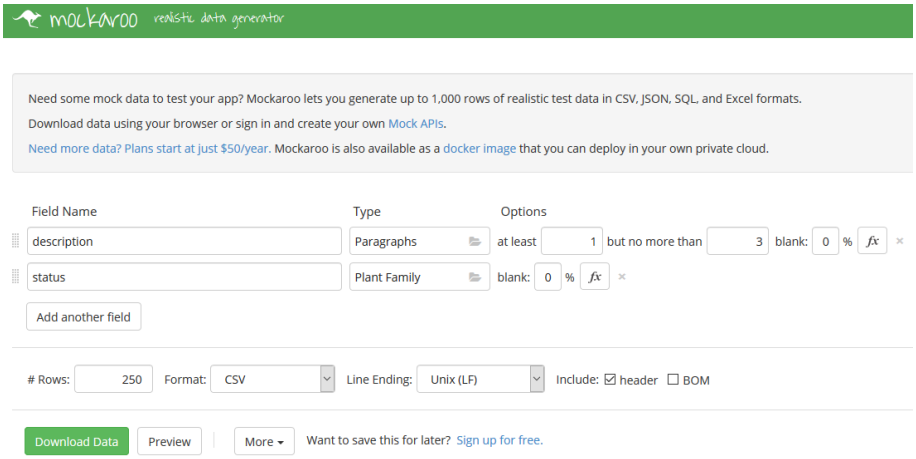


Figure 135 Injecting values with Moockaro

We chose the US-003 (Medical Record Creation) for which the positive scenario was to create a valid medical record with its description and status, and a negative case scenario which comes from trying to create a medical record without description or status. I created 250 samples for each positive and negative case scenario.

Then we modified the existent scala file un order to use the CSV.

```
object MedicalRecordFormPositive {
  val feederPositive = csv("feederPositive.csv")
  val medicalrecordformpositive = exec(http("medicalrecordformpositive")
                .get("/owners/6/pets/8/visits/3/medical-record/new")
                .check(css("input[name=_csrf]","value").saveAs("stoken")))
        .pause(24)
        .feed(feederPositive)
        .exec(http("creation")
                .post("/owners/6/pets/8/visits/3/medical-record/new")
                .headers(headers_2)
                .formParam("id", "")
                .formParam("description", "${description}")
                .formParam("status", "${status}")
                .formParam("_csrf", "${stoken}"))
        .pause(11)
}

object MedicalRecordFormNegative {
  val feederNegative = csv("feederNegative.csv")
  val medicalrecordformnegative = exec(http("medicalrecordformnegative")
                .get("/owners/6/pets/8/visits/3/medical-record/new")
                .check(css("input[name=_csrf]","value").saveAs("stoken")))
        .pause(24)
        .feed(feederNegative)
        .exec(http("error")
                .post("/owners/6/pets/8/visits/3/medical-record/new")
                .headers(headers_2)
                .formParam("id", "")
                .formParam("description", "${description}")
                .formParam("status", "${status}")
                .formParam("_csrf", "${stoken}"))
        .pause(11)
}
```

Figure 136 Changes made to Scala

After that, we ran the script and look at the database. As we can see, the 250 new medical records that were declared in the CSV file were correctly added.

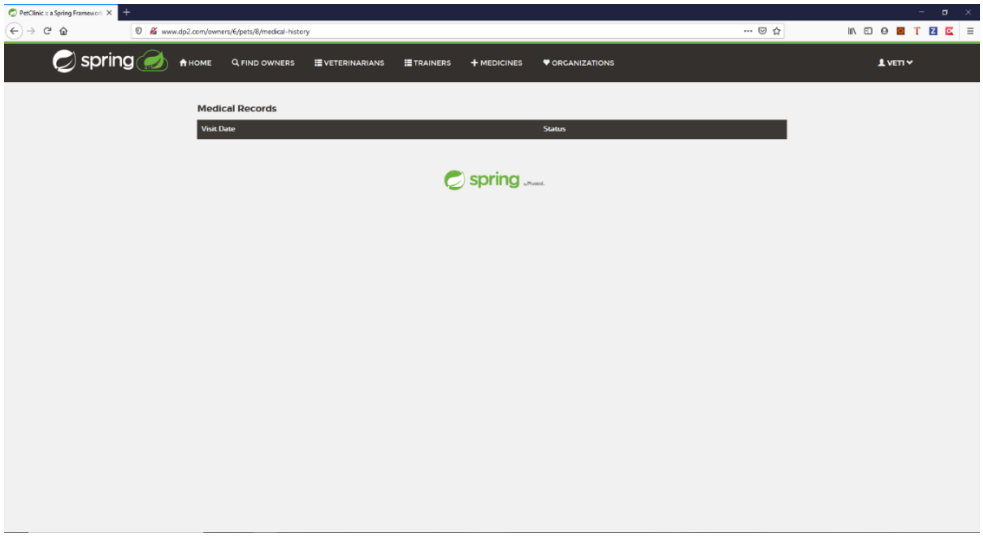


Figure 137 Listing before executing the script

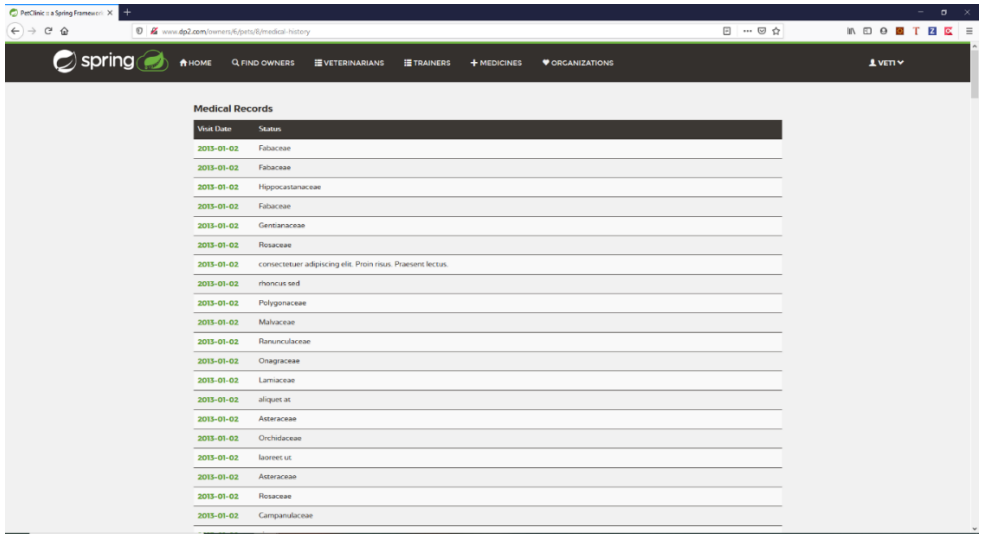


Figure 138 Listing after executing the script

And we enter the view to check that it was created correctly

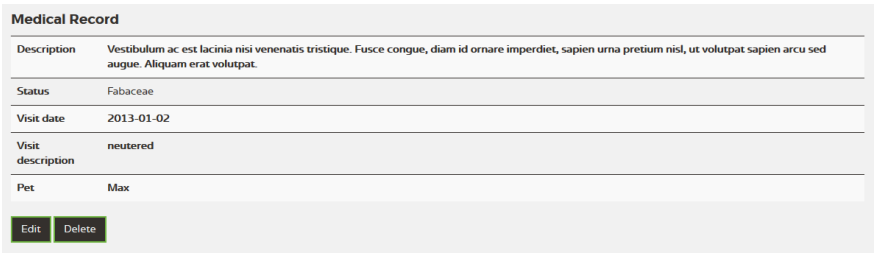


Figure 139 View example