

--	--	--

**Álvaro Rubia Tapia**

**Study of the use of feeders in Gatling:**

<b>Introduction</b>	<b>2</b>
<b>Scala file</b>	<b>3</b>
Definition of values and headers	3
Objects	3
Scenarios	3
Positive	4
Negative	4
SetUp	4
<b>CSV files</b>	<b>5</b>
<b>Result</b>	<b>6</b>

		1
--	--	---

## Introduction

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.



```
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
```

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 gatlin, It is known that a scala file is necessary to run it, that will be saved in the simulation folder, where as the files that we are going to use (csv format) will be contain in the resources file, so once gatlin 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 gatlin, so there is little work to do, just revise the blacklist.

The other headers are also created automatically.

```
1 package dp2
2
3 import scala.concurrent.duration._
4
5 import io.gatling.core.Predef._
6 import io.gatling.http.Predef._
7 import io.gatling.jdbc.Predef._
8
9 class InterventionCreateDiagnosis extends Simulation {
10
11   val httpProtocol = http
12     .baseUrl("http://www.dp2.com")
13     .inferAllResources(BlackList(""".*ico""", """.*png""", """.*js""", """.*css"""), Whitelist())
14     .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")
15     .acceptEncodingHeader("gzip, deflate")
16     .acceptLanguageHeader("es-ES;q=0.9,en;q=0.8")
17     .upgradeInsecureRequestsHeader("1")
18     .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36")
19
20   val headers_0 = Map("Proxy-Connection" -> "keep-alive")
21
22   val headers_2 = Map(
23     "Origin" -> "http://www.dp2.com",
24     "Proxy-Connection" -> "keep-alive")
25 }
```

### 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)  
}
```

### 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)  
}
```

### SetUp

Inside the setUp part we have two things.

The first one is the code that executes the scenarios when the file is runned. 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)))
.protocol(httpProtocol)
.assertions(
global.responseTime.max.lt(5000),
global.responseTime.mean.lt(1000),
global.successfulRequests.percent.gt(95))
```

## CSV files

The csv files have been done using [www.mockaroo.com](http://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.

The screenshot shows the Mockaroo website interface for generating CSV data. The header includes the Mockaroo logo, a 'realistic data generator' tagline, and links for 'PRICING' and 'SIGN IN'. Below the header, there is a message about downloading data and creating Mock APIs, with a link to 'Need more data?'. The main configuration area has three columns: 'Field Name', 'Type', and 'Options'. Three fields are defined: 'intervention\_date' (Datetime, range 07/31/2019 to 09/30/2020), 'intervention\_time' (Number, range 1 to 8), and 'intervention\_descriptic' (Sentences, range 1 to 10). Below these fields is a button 'Add another field'. At the bottom, there are settings for '# Rows' (200), 'Format' (CSV), 'Line Ending' (Unix (LF)), and 'Include' (header checked, BOM unchecked). There are buttons for 'Download Data', 'Preview', and 'More', along with a link to 'Sign up for free'.

		5
--	--	---

--	--	--

Negative scenario:

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

The image shows the Mockaroo website interface for generating data. The header includes the Mockaroo logo, 'realistic data generator', and links for 'PRICING' and 'SIGN IN'. Below the header, there is a green banner with instructions: 'Download data using your browser or sign in and create your own Mock APIs.' and 'Need more data? Plans start at just \$50/year. Mockaroo is also available as a docker image that you can deploy in your own private cloud.'

The main form is titled 'Field Name', 'Type', and 'Options'. It contains three rows of data fields:

- intervention\_date**: Datetime, range from 07/31/2019 to 09/30/2020, format yyyy-mm-dd, blank: 80%, fx.
- intervention\_time**: Number, min: 1, max: 8, decimals: 0, blank: 80%, fx.
- intervention\_descriptic**: Sentences, at least 1 but no more than 10, blank: 80%, fx.

Below the fields, there is a button 'Add another field'. At the bottom, there are settings for '# Rows: 200', 'Format: CSV', 'Line Ending: Unix (LF)', and 'Include: ☒ header ☐ BOM'. There is a green 'Download Data' button, a 'Preview' button, and a 'More' button. A link 'Want to save this for later? Sign up for free.' is also present.

## Result

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

The image shows a Windows command prompt window with the title 'C:\WINDOWS\system32\cmd.exe'. The output of the command is as follows:

```

[3] computerdatabase.advanced.AdvancedSimulationStep03
[4] computerdatabase.advanced.AdvancedSimulationStep04
[5] computerdatabase.advanced.AdvancedSimulationStep05
[6] dp2.BasicExample
[7] dp2.InterventionCreateDiagnosis
[8] dp2.us14
[9] dp2.us15
[10] dp2.us16
[11] dp2.us17
[12] dp2.us20
7
Select run description (optional)
Simulation dp2.InterventionCreateDiagnosis started...

-----
2020-05-29 16:03:42                               5s elapsed
-----
Requests -----
> Global (OK=8 KO=0 )
> Home (OK=8 KO=0 )

-----
---- InterventionCreationPositive -----
[-----] 0%
waiting: 36 / active: 4 / done: 0
---- InterventionCreationNegative -----
[-----] 0%
waiting: 36 / active: 4 / done: 0
-----

```

The window also shows the Windows taskbar at the bottom with the search bar and system tray.

		6
--	--	---

--	--	--

```

C:\WINDOWS\system32\cmd.exe
> loggedRedirect 1 (OK=80 KO=0 )
> InterventionForm (OK=40 KO=0 )
> InterventionFormPositive (OK=40 KO=0 )
> InterventionCreationFormWithErrorMessage (OK=40 KO=0 )
> InterventionCreated (OK=40 KO=0 )

----- InterventionCreationPositive -----]100%
waiting: 0 / active: 0 / done: 40
----- InterventionCreationNegative -----]100%
waiting: 0 / active: 0 / done: 40

Simulation dp2.InterventionCreateDiagnosis completed in 147 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

----- Global Information -----
> request count 480 (OK=480 KO=0 )
> min response time 4 (OK=4 KO=0 )
> max response time 2478 (OK=2478 KO=0 )
> mean response time 59 (OK=59 KO=0 )
> std deviation 204 (OK=204 KO=0 )
> response time 50th percentile 28 (OK=28 KO=0 )
> response time 75th percentile 51 (OK=51 KO=0 )
> response time 95th percentile 118 (OK=118 KO=0 )
> response time 99th percentile 891 (OK=891 KO=0 )
> mean requests/sec 3.243 (OK=3.243 KO=0 )

----- Response Time Distribution -----
> t < 800 ms 474 ( 95%)
> 800 ms < t < 1200 ms 2 ( 0%)
> t > 1200 ms 4 ( 1%)
> failed 0 ( 0%)

Reports generated in 1s.
Please open the following file: C:\Users\VALVARO\gating-charts-highcharts-bundle-3.3.1\results\interventioncreatediagnosis-20200529140333260\index.html
Global: max of response time is less than 5000.0 : true
Global: mean of response time is less than 1000.0 : true
Global: percentage of successful events is greater than 95.0 : true
Presione una tecla para continuar . . .

```

The result of Gatlin are saved in a result file. Here we have the statistics of how did the process go.

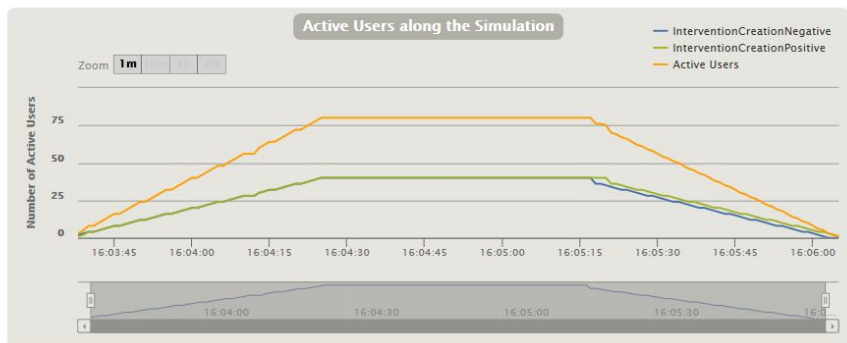


As we can see here, the assertions defined have status OK.

		7
--	--	---

--	--	--

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	480	480	0	0%	3.243	4	28	51	118	891	2478	59	204	
Home	80	80	0	0%	0.541	15	41	58	120	223	223	51	42	
Login	80	80	0	0%	0.541	4	9	13	27	31	34	11	6	
logged	80	80	0	0%	0.541	5	11	16	28	47	50	13	8	
logged Redirect 1	80	80	0	0%	0.541	8	19	27	57	90	95	25	18	
Interven...Positive	40	40	0	0%	0.27	29	76	116	926	2204	2478	202	467	
InterventionForm	40	40	0	0%	0.27	24	65	106	976	2228	2476	196	476	
Interven...rMessage	40	40	0	0%	0.27	25	45	58	105	129	143	53	27	
InterventionCreated	40	40	0	0%	0.27	24	44	55	107	181	198	52	33	



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.