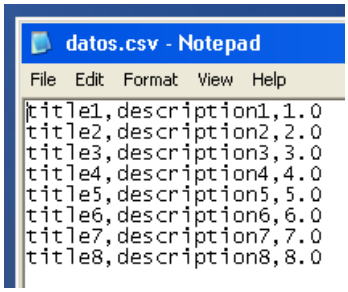
DELIVERABLE 10 ITEM 7: USING CSV FILES IN JMETER

In this deliverable, we are testing the performance of our project using a tool JMeter. This tool is very useful for this purpose and it is also very easy to set up and use. In order to start testing a use case, we had to configure a proxy in JMeter so that it could keep track of the different URLs that we were using in our browser, and, that way, generate a list of requests to be made to our web site to test its performance. While using the proxy, the URLs are not the only thing being tracked and recorded; JMeter also stores the different values that are sent in a request to the server. This includes every POST request, that is, every one related to forms. At first glance this can be seen as something really useful, as we do not need to do barely anything to develop our tests. However, there is an important drawback in this process. While JMeter records a test script, we only complete each relevant form only once. This means that the program stores only one value for each form field. So, when we execute our tests, which will fill each form a thousand or more times, will always use the same values once and again. For some tests, this is not really relevant, as creating several objects with the same attributes is not problematic. However, there are some cases where this is a problem, and it would be convenient to be able to specify a list of different values and allow JMeter to pick them. Fortunately, JMeter supports the use of CSV files to allow the use of dynamic data in our tests. CSV is the short form of "Comma Separated Values", and a file with this format consists of several lines containing lists of values, separated by commas. This format is widely used because it is easy to handle by a computer, and some tables can be easily represented in a CSV file. Another great advantage of this format is that it is simple enough to be edited in any text editor, including, for example, Notepad.

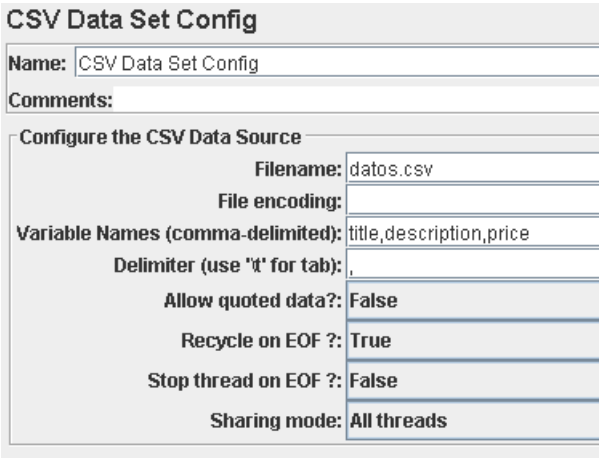
For this example, we will be testing the use case that involves the creation of a newspaper. The full test case consists of the user accessing the main page of the web site, logging in with their account, and creating the new newspaper. The user that will be logged in is "user1", and this will not change in any test. The form that will be used to test a CSV file will be the one involving the creation of the newspaper. To be more precise, the values of the newspaper to be included in the CSV will be the title, the description and the price. The picture is not necessary, so it is better to leave it blank to keep the CSV a little bit simpler. The file that we will use is the following one:



For this case, eight different sets of values are enough. For the sake of simplicity, we used the same number for the title, the description and the price. This way, we can check that JMeter is introducing the correct data into the database by accessing to the web site after the test has finished its execution.

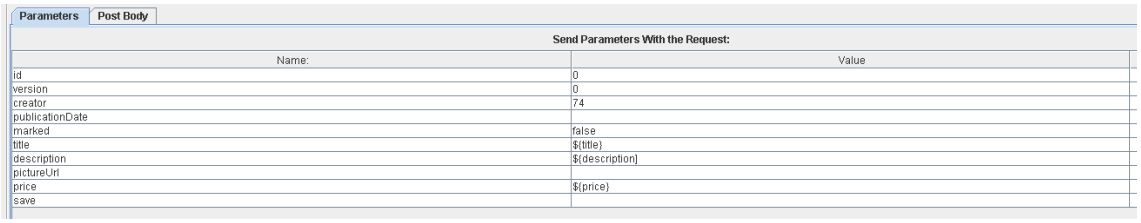
To specify that we are going to use a CSV file in JMeter, we will open the program, right click the test plan that we want to use, select "Add", then "Config Element", and finally click on "CSV Data Set Config". This will open a view where we can configure the different parameters

of the CSV. We will explain them briefly, to be aware of the interesting parameters that we can change. The following screenshot shows how this view is structured (note that we have cropped the image, as nothing else to the sides of the screenshot was relevant):



The first and second fields are already familiar to us from the other configurations that we had to do in JMeter: a name for the element and some comments. We left these fields with their default values. The first important parameter is the “Filename”. To be able to directly specify its name, we must include the CSV file in the same folder where the JMX file (the one that contains the test itself) is located. In “File encoding” we can specify the character set encoding; we can leave this field blank. Then we need to choose the variable names. We can write whatever names we prefer, as these names do not need to be the same as the names of the form fields. We can also specify which delimiter our file is using to separate the different variables. Since we are using a CSV file, the most logical delimiter is a comma (which is the default value), but we could specify something different if we want to include a comma in any value. We can also specify if we want to recycle (that is, start over from the beginning) once the tests reach the end of the file; we will leave this option set to “true”. We can also tell JMeter if we want to stop a thread when the end of the file is reached; we set this to “false”. Finally, we can specify if we want all threads to share the CSV file, or to have separate files for each of them; we will use only one file for all threads.

After these configuration parameters are set, we need to find which entry in the test script contains the POST request that will use the CSV file. In this case, the request is “/newspaper/user/edit.do”. To tell JMeter that a parameter must be taken from a CSV, we use a notation similar to that of JSP: ${name}, where “name” is one of the variable names that we specified in the CSV configuration. For our example, we set the different values as shown in this screenshot:



Once the values are set, we can execute the test. If everything is working correctly, we can query the database or go to the web site and see that all the newspapers have been created, using the eight different sets of values that we defined in the CSV. And with these easy steps, we have created a simple dynamic test.

Finally, we are going to analyze how the test turned out. We ran this test on a computer with the following specs:

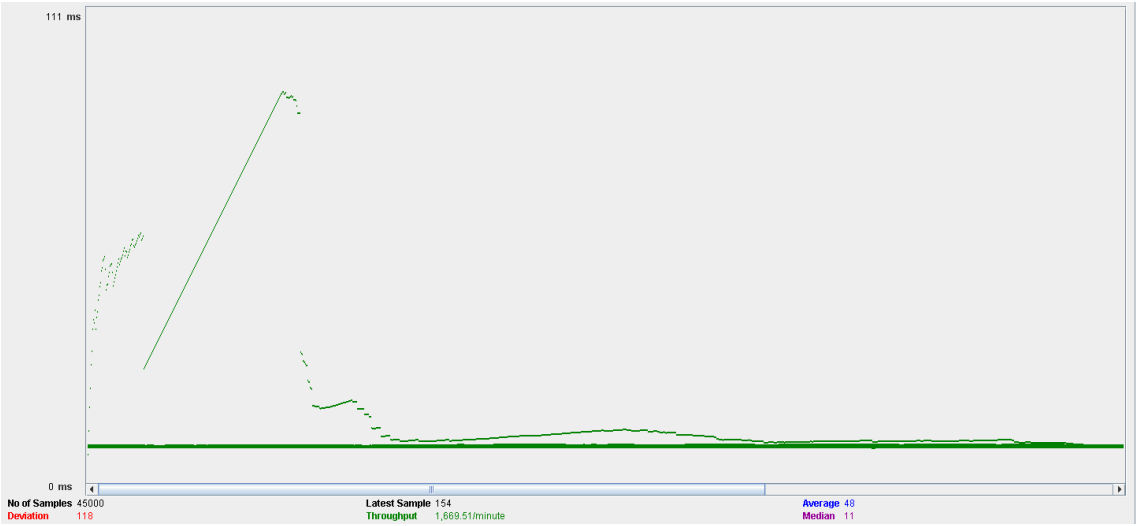
Processor: Intel(R) Core(TM) i7-7700HQ (2.8 GHz, Turbo Boost up to 3.8 GHz, 6 MB cache, 4 cores)

RAM: SDRAM 8 GB DDR4-2400 (2x4 GB)

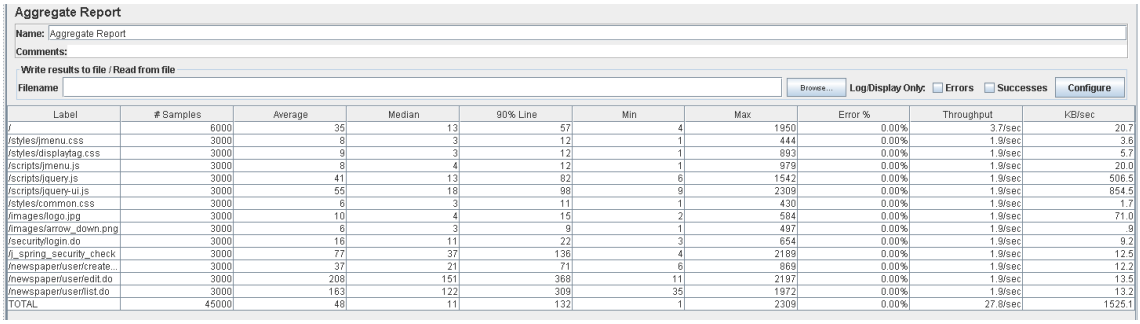
HDD: 1 TB SATA 7200 rpm

Network Adapter: Intel(R) Dual Band Wireless-AC 7265

These are the results of the test, ran with 30 concurrent users, a ramp-up period of 1 second, and 100 loops:

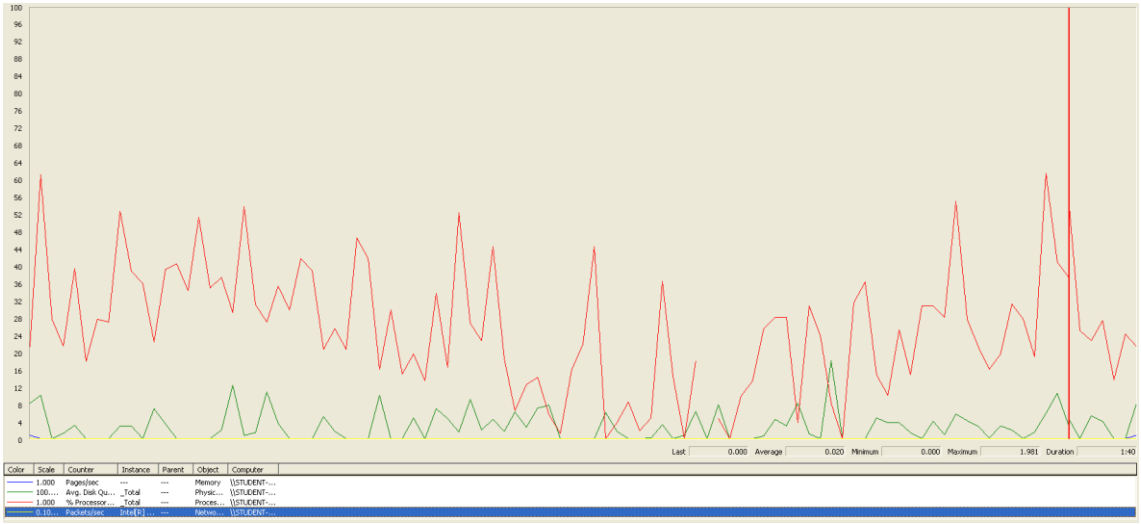


This is the aggregate report:



Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
/	6000	35	13	57	4	1950	0.00%	3.7/sec	20.7
/styles/jmenu.css	3000	8	3	12	1	444	0.00%	1.9/sec	3.6
/styles/displaytag.css	3000	9	3	12	1	893	0.00%	1.9/sec	5.7
/scripts/jmenu.js	3000	8	4	12	1	979	0.00%	1.9/sec	20.0
/scripts/jquery.js	3000	41	13	82	6	1542	0.00%	1.9/sec	506.5
/scripts/jquery-ui.js	3000	55	18	98	9	2309	0.00%	1.9/sec	854.5
/styles/common.css	3000	6	3	11	1	430	0.00%	1.9/sec	1.7
/images/logo.jpg	3000	10	4	15	2	584	0.00%	1.9/sec	71.0
/images/arrow_down.png	3000	6	3	9	1	497	0.00%	1.9/sec	.9
/security/login.do	3000	16	11	22	3	654	0.00%	1.9/sec	9.2
/j_spring_security_check	3000	77	37	136	4	2189	0.00%	1.9/sec	12.5
/newspaper/user/create...	3000	37	21	71	6	869	0.00%	1.9/sec	12.2
/newspaper/user/edit.do	3000	208	151	368	11	2197	0.00%	1.9/sec	13.5
/newspaper/user/list.do	3000	163	122	309	35	1972	0.00%	1.9/sec	13.2
TOTAL	45000	48	11	132	1	2309	0.00%	27.6/sec	1525.1

Here, we can see that the time that the web site takes to fully complete this use case is approximately 1.346 seconds for the 90 % of the users. This is a pretty acceptable time, taking into account that at the end of the test we already have around 3000 newspapers in the database. We can say that our system can easily handle 30 concurrent users for this use case. We will analyze the performance of the computer during the test. This is the graph:



As we can see, the computer did not experience a heavy load in any moment during the execution. The processor was the component under the biggest stress, and it may become a bottleneck if we further increase the number of concurrent users.