Revista Romana de Interactiune Om-Calculator 9(2) 2016, 155-176 © MatrixRom

Methodology for Identification and Evaluation of Web Application Performance Oriented Usability Issues

Mihaela Ciugudean, Dorian Gorgan

Computer Science Department, Technical University of Cluj-Napoca
Str. G. Baritiu, 28, 400027, Cluj-Napoca
E-mail: ciugudeanmihaela@yahoo.com, dorian.gorgan@cs.utcluj.ro

Abstract. This paper aims to illustrate a methodology for identifying and assessing a set of performance issues encountered in a particular web application, with impact on the usability level. Throughout this methodology, several visual techniques are determined and investigated by taking advantage of the functionalities offered by available performance monitoring tools such as JMeter and VisualVM. This way, the major performance concerns and their impact on the web application usability are easily determined. Therefore, by identifying the problematic parts of a system, corrective measures could be taken, particularly designed to operate on the root cause of the problem, thus leading to the targeted objective. Moreover, some performance improvement recommendations are further on presented in order to enhance the overall usability and the user experience, the actual goal of the research and development activities conducted. These performance concerns might be designed and implemented after fixing the critical parts causing usability problems and consequently, maximizing the user satisfaction and comfort when exploring the functionalities offered by a particular web application.

Keywords: Interactive software application; usability; performance improvement; JMeter; VisualVM; user experience; visual techniques.

1. Introduction

Over the last past decades, the web has evolved into a growing universe of interconnected web pages and applications, offering the end user a wide variety of services and functionalities. With the web development, the interactive web applications have become easily accessible regardless of place and time concerns and without any installation requirements (Rossi et al., 2007). As a consequence, end users have begun to favor web applications over traditional desktop applications. On the other hand, the rapidly increasing mobile application market has shown that web applications can be easily installed on devices.

In particular, the web enables marketers to get in touch with the visitors of their websites or applications and start communicating with them. Furthermore, the web might be perceived as an excellent sales channel for millions of applications, either large or small.

More precisely, end users take advantage of the wide variety of tools and services offered by web applications, by means of web browsers. Therefore, the customers have the opportunity to access, retrieve and interact with data and content located on different software applications available over the web, as stated by Mohler (Mohler and Duff, 2000).

However, due to the fast growth of the web, an intensive competition has been created among web applications. According to Stoneham and Dastbaz (2006), the web, perceived as an application platform, has raised the standard for the provided web applications, establishing a norm for highly-interactive dynamic user interfaces with real time collaborative features. In case a certain web application fails to completely fulfill the customer's needs and gets him unsatisfied, he will start using services offered by a similar website. Therefore, application developers have begun to enhance the accessibility, usability and user experience when providing specific services and products in order to meet end users' demands.

When creating a web application, it is not sufficient to provide an intuitive, self-explanatory user interface that will allow even an unexperienced computer user to perform the desired actions. As specified by Shivakumar (2014), designing a consistent and standardized UI may still not satisfy the customer, determining him to abandon your website and switch to a similar one that completely fulfills his demands. All this frustration and annoyance that might lead a user to discard your application is very often caused by a series of usability problems occurring in the website.

Considered one of the most important quality factors for Web applications, usability has been receiving great attention, being recognized as a fundamental property for the success of web applications (Barish, 2002). As previously said, it is not sufficient to satisfy the functional requirements of a web application in order to ensure its success. According to Purba (2001), the ease or difficulty experienced by users of these web applications is largely responsible for determining their success or failure. Consequently, usability evaluations and technologies that support the usability design process have therefore become critical in ensuring the success of web applications.

The scalability should be an important characteristic of the web application. The main requirement on the usability concerns with keeping a high level of usability at any level of scalable application. It is not enough to check the usability level for single user, but to check usability of the application for a

great or huge number of users.
The aim of this study is to present the way by which the visual techniques allow to recognize the root cause of the usability issues encountered in a scalable web application, thus allowing for immediate identification of the technical reason. The paper presents a way of highlighting the usability issues and a direct causal link to the corespondent technical issues.
By taking advantage of the various performance indicators when automatically testing a web application, a particularized methodology might be applied in order to remove, or at least reduce, the encountered system's weaknesses. This way, time and effort invested in unsuccessful operations to improve the overall usability of a web application are saved by starting to design and implement a solution centered on the specific issue that acts as a bottleneck in the system. By means of technical and technological solutions implemented to fix the determined root cause, the overall application usability will be enhanced.

The rest of this paper proceeds as follows. In Section 2, state of the art is presented and the contribution of the current work is highlighted. Section 3 details the proposed methodology by presenting and exemplifying each step. Finally, some concluding remarks and recommendations are proposed.

2. Related Works

The exponential growth of the internet and of web applications leads to the necessity to evaluate them from a quantitative point of view. In the past years, valuable methodologies have been used to evaluate the quality of specific web applications and discover possible issues. More precisely, inspection methods such as Cognitive Walkthrough for the Web (CWW) and Web Design Perspectives (WDP) are proposed by Haralambos and Colette (2011) to identify usability problems. According to them, the aforementioned methods are characterized by a high degree of subjectivity in usability evaluations. In order to overcome this drawback, these methods are sometimes replaced by inspection metrics-based ones which are likely to reduce the subjectivity degree, such as WebTango and Web Quality Evaluation Method (WebQEM). More precisely, WebTango provides quantitative metrics, based on empirically validated metrics for user interfaces to build predictive models in order to evaluate other user interfaces.

On the other hand, WebQEM performs a quantitative evaluation of usability aspects, aggregating them to obtain usability indicators. As stated by Singh (2016), the purpose of this method is to systematically assess characteristics, sub-characteristics and attributes that influence products' quality, finally yielding to global, partial and elementary quality indicators that can help different stakeholders in understanding and improving the assessed product. According to Olsina and Rossi (1999), by implementing the four major technical steps of WebQEM methodology, it can be employed in assessing and comparing quality requirements in the operative phase of Web sites and applications as well as in early phases of Web development projects. By using the methodology, either absent attributes, absent sub-characteristics, or requirements poorly implemented might be easily discovered. As concluding by Mendes and Mosley (2006), WebQEM can be used to assess diverse application domains according to different user views and evaluation goals.

Nowadays, more and more companies are concerned with addressing usability aspects when designing their website. This concern has developed as a consequence of the competitive market in which each organization strives to continuously enlarge the variety of users by providing an intuitive, self-explanatory and accessible website. Usability is an even broader goal than accessibility, which refers to how easily a website can be used, understood, and even accessed by people with disabilities.

Starting with 2015, even Google has started to evaluate the usability of web applications, mostly their mobile version, and announced by email their owners on the poor rated usability results. Moreover, there was included a list of aspects to be addressed in order to overcome these usability concerns.

One example of receiver of such an acknowledgement was WordPress, which begun to assess the concerns presented in the email received from Google and check their behavior in the mobile version of the web application.

A similar case study of a website trying to enhance its usability and accessibility features is The American Foundation for the Blind. Their strategy was to undergo a major redesign in order to improve the site's usability and create a more logical, user-friendly information architecture.

However, these are only two examples of websites dealing with usability and accessibility concerns in order to maximize the user experience, maintain and enlarge the number of customers. Many other web applications have realized that a consistent and intuitive user interface is sometimes not enough to please the customer's needs and therefore, started to analyze and fix their usability issues.

3. Methodology for Identifying and Solving Usability Issues

The purpose of this paper is to briefly describe the methodology used for identifying the usability issues in a specific web application and inferring the technical reason. Therefore, the considered context is for instance, the following one: starting from a particular web application, its usability is defined as the system' capability to successfully handle a variable number of concurrent users. This case the usability is evaluated by the reported error percentage for each case scenario. Therefore, the main step is to measure the application's responsiveness-oriented behavior when successively increasing the number of users, by taking advantage of the functionalities offered by JMeter tool (JMeter, 2016). After concluding the existence of such usability issues, their root cause is to be determined by means of VisualVM performance monitoring tool (VisualVM, 2016), thus obtaining accurate information regarding the sources of the problem centered on two diagnosis indicators (CPU and Memory consumption). The further step is to analyze the situation, propose and implement specific solutions to overcome the usability problem. Finally, by measuring the application's behavior after applying the established solution, the followed methodology is validated and conclusions related to its accuracy may be stated.

3.1 Usability

Being one relevant component of web applications' quality, defined as the extent to which a system is able to satisfy its customers in efficiently achieving their goals, usability is recognized as a fundamental property for software applications' success. In the context of software lifecycle, usability is perceived as being relevant to all its stages, not only at the end of the product development.

According to Olsina and Rossi (1999), a software product quality might be defined in terms of fundamental characteristics (usability, functionality, reliability, efficiency, portability, and maintainability) as defined in the ISO/IEC 9126-1 standard, usability being one of them.

Specifically, for the considered web application, usability metrics are assessed by monitoring the system's capacity to handle a reasonable amount of concurrent usage. More precisely, the Application Performance Monitoring tests are executed in the context of successively increasing the workload expressed as the number of concurrent active users and monitoring the system's behavior. Therefore, the exemplified attribute in usability evaluation is the number of concurrent users that is successively increased and obtained results interpreted from the reported error percentage values.

3.2 Use case scenario: Online Scrum tool

To begin with, the web application under analysis represents a system capable of providing a tool used as an agile project organizer. More precisely, the application is intended to offer the functionalities of an online tool which allows the persons involved in the Scrum process to perform it regardless of any inconveniences such as physically distributed teams, unreachable customer or any other factors that may negatively affect the Scrum process and thus the corresponding software project, (Rubin, 2013) and (Schwaber and Sutherland, 2013).

In this context, one use case that has been studied mainly consists of the following actions to be undergo by a privileged user, say a Scrum Master: login into the application, visualize all developers, then visualize the taskboard, add a new task (together with the required information for it), visit again the list of developers and finally the log out from the application.

3.3 Steps of identifying and solving usability issues

In order to identify and evaluate the issues encountered in a web application, by means of visual techniques offered by performance monitoring tools such as JMeter and VisualVM, several use case scenarios were considered. Consequently, for these use cases, a series of experiments were conducted and the visual results were analyzed so that to determine the root causes of the usability issues and propose technical solutions to successfully solve them.

We may define visual techniques as the ensemble of differently shaped graphical representations of the executed testplans that allow evaluation based on various diagnosis indicators while adjusting the variable parameters and performing successive measurements. These visual techniques apply in the context of using dedicated Application Performance Monitoring (APM) tools, such as JMeter and VisualVM, used as a basis in building the methodology for detecting potential performance weaknesses of the system under test and subsequently propose particular solutions to overcome them.

Step 1: Determine the existence of usability problems

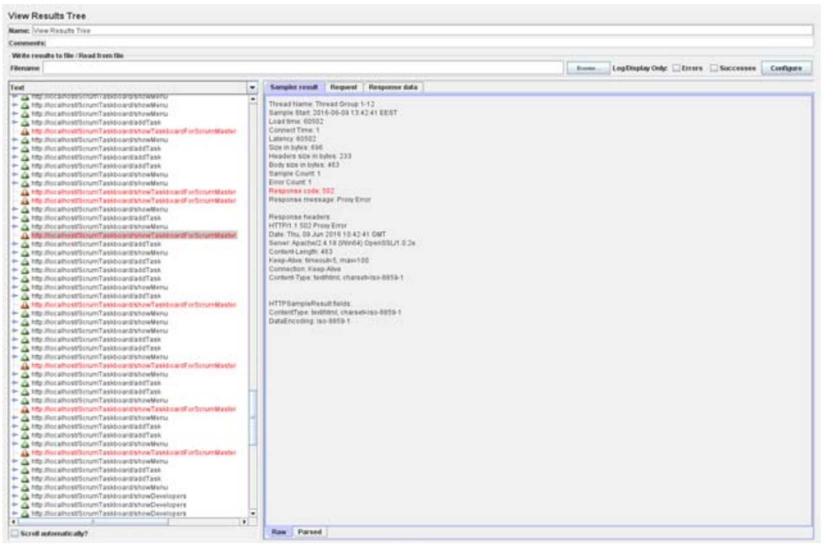
In order to assess non-functional system's characteristics such as usability, scalability, availability, scalability, response time, latency, throughput and many other performance oriented metrics, JMeter monitoring tool was firstly used and the reported results analyzed by means of different visual techniques (JMeter, 2016).

The aim is to determine potential usability issues in the context of a high workload obtained from a large number of concurrent users that access the web application. The basic idea is that some usability issues are not revealed at low scale of application. The goal is to identify the context in which such issues become detectable. It is difficult and sometimes almost impossible to manually perform all these performance tests, by considering a large number of online users. As a result, these cases were performed by means of the JMeter tool which automatically executes the desired scenarios by simulating the existence of the desired number of users.

The first step of the proposed methodology was to conduct JMeter performance tests, simulating a variable number of users that simultaneously execute this use case. In the followed context, the considered web application as well as all the system's configuration are the fixed elements, while the number of concurrent users represents the variable element. Therefore, specifically for the considered use case, the evaluation is performed by analyzing the reported error by successively increasing the number of concurrent users.

A key point to mention is that the use case scenario under test is characterized by accessing of large amount of data: a list of thousand developer instances which is twice retrieved from the database. As a result, it is intended to illustrate the web application's behavior when operating on large set of data, context specific to mostly all real life production environments. However, in order to perform automatic monitoring and testing of the application's performance, the number of concurrent users (configured in the corresponding JMeter testplan) was selected in the range of tens. The main reason behind this decision is represented by the considerable amount of time required to execute and collect results in the context of an increased number of concurrent threads, which would be outside the scope of testing a small-scaled web application.

Finally, the measurable attributes are the reported error percentage, standard deviation, throughput, response time, number of KB/sec, latency, bandwidth etc.



```
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/addTask
http://localhost/ScrumTaskboard/showMenu
http://localhost/ScrumTaskboard/showDevelopers
http://localhost/ScrumTaskboard/showDevelopers
http://localhost/ScrumTaskboard/showDevelopers
```

```
Thread Name: Thread Group 1-12
Sample Start: 2016-06-09 13:42:41 EEST
Load time: 60502
Connect Time: 1
Latency: 60502
Size in bytes: 696
Headers size in bytes: 233
Body size in bytes: 463
Sample Count: 1
Error Count: 1
Response code: 502
Response message: Proxy Error

Response headers:
HTTP/1.1 502 Proxy Error
Date: Thu, 09 Jun 2016 10:42:41 GMT
Server: Apache/2.4.18 (Win64) OpenSSL/1.0.2e
Content-Length: 463
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1


HTTPSampleResult fields:
ContentType: text/html; charset=iso-8859-1
DataEncoding: iso-8859-1
```

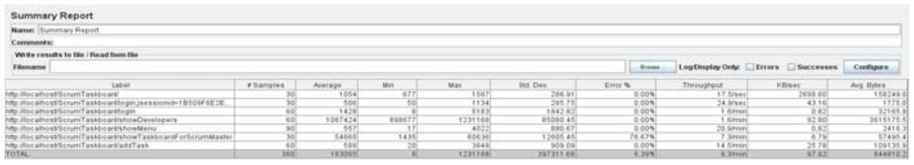
Figure 1. View Results Tree – 30 users.

Implementing the first step of the proposed methodology means following one of the two branches, depending on the measurable element: percentage error and throughput value. Therefore, the presence of usability problems might be determined based on the two measurable elements.

Reported percentage error

To begin with, for 10 concurrent threads which simultaneously execute the sample case scenario (login into the application, visualize all developers, then visualize the task board, add a new task - together with the required information for it, visit again the list of developers and finally the log out from the application), the application successful handles them all, fact denoted by the reported 0% error.

However, if the number of concurrent threads is increasing up to 30, the application responds with error for some of these 30 HTTP requests. This result might be visualized from the View Results Tree image (which was obtained from JMeter tests) in which the output of each of the 30 user requests to the application (HTTP requests) is displayed, as well as relevant information for all of them (Figure 1). The precise error percentage that was reported for this case scenario is 6.39%, as illustrated in the Summary Report (Figure 2).



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: Errors Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
http://localhost/ScrumTaskboard/	30	1054	677	1587	286.91	0.00%	17.5/sec	2698.60	158249.0
http://localhost/ScrumTaskboard/login;jsessionid=1B509F6E2E...	30	506	50	1134	285.75	0.00%	24.9/sec	43.16	1775.0
http://localhost/ScrumTaskboard/login	60	1428	8	5183	1842.82	0.00%	1.6/min	0.82	32165.9
http://localhost/ScrumTaskboard/showDevelopers	60	1067424	898677	1231168	85080.45	0.00%	1.6/min	92.80	3615575.5
http://localhost/ScrumTaskboard/showMenu	90	557	17	4022	890.67	0.00%	20.9/min	0.82	2416.3
http://localhost/ScrumTaskboard/showTaskboardForScrumMaster	30	54665	1435	60636	12605.45	76.67%	7.3/min	6.79	57495.4
http://localhost/ScrumTaskboard/addTask	60	588	20	3648	909.09	0.00%	14.5/min	25.78	109135.9
TOTAL	360	193065	8	1231168	397311.68	6.39%	9.3/min	97.92	644810.2

Figure 2. Summary report – 30 users.

As the particular web application is intended to serve a large number of concurrent online users, it has to successfully respond to all of them, or to as much as possible. Therefore, the purpose of this step was to assess the system’s behavior in the context of simulating a large users’ number. If these results seem worrying for a number of 30 users concurrently accessing the system, the application performs even worse when increasing the number of users in the range of hundreds. This outcome was determined by performing the same JMeter performance tests with an increased number of 100 users and evaluating the results. This case was almost impossible to test as increasing the users’ number lead to an unresponsive application, thus denoting the severe scalability problems faced by the web application under test.

Consequently, by means of these visual techniques and of the measured percentage error as a performance indicator, the proposed methodology illustrates the system’s behavior evaluated in the context of subsequently changing the number of users concurrently accessing the application.

Reported throughput value

When identifying the existence of usability problems in a web application by using the throughput as measurable element, decision-related aspects have to be considered regarding the significance of this performance indicator and the way to interpret its value. Therefore, we define throughput as a non-functional, performance indicator requirement, measured as the total number of requests (transactions) in a given time. It is usually referenced in the literature as TPS (transaction per second).

Being perceived as a significant indicator that helps in evaluating the performance of an application, the throughput reflects the capacity of the server, how much load it can take. Even though maximum throughput is desirable, a system’s performance should be evaluated based on several other indicators such as response time, latency, bandwidth, etc. In our web application context, throughput is measured as the number of requests sent

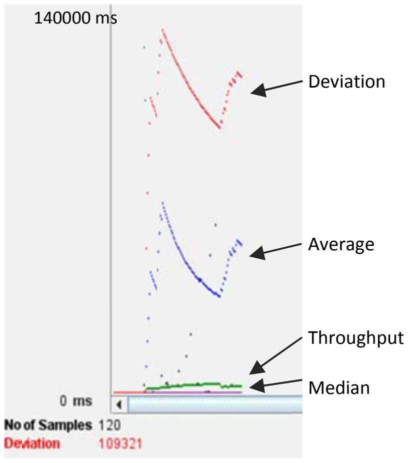


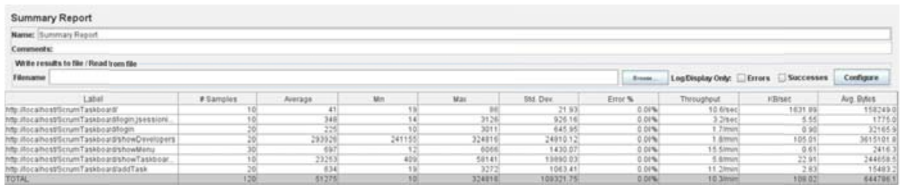
Figure 3. Graph results – 10 users.

to the web server per second.

In assessing the throughput of a web application, JMeter provides relevant results by means of several listener components. Specifically, for the tested use case scenario, the reported throughput was determined by analyzing the output of Graph Results (Figure 3) and Summary Report (Figure 4) listeners. Also, the performance tests were executed under the same context as for the error-based approach, by using the number of users as variable element and selecting 10 and 30 as reference values.

For 10 threads that simultaneously execute the sample use case scenario, the throughput obtained from these listeners is presented in Figures 3 and 4.

While the Graph Results listener (Figure 3) specifies the resulted throughput of an executed test, the Summary Report one (Figure 4) offers a more detailed view of this measurement corresponding to each HTTP request in the considered use case scenario. Therefore, we may easily


Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
http://localhost/ScrumTaskboard/	10	41	19	86	21.93	0.00%	10.6/sec	1631.89	158249.0
http://localhost/ScrumTaskboard/login;jsessioni...	10	348	14	3126	926.16	0.00%	3.2/sec	5.55	1775.0
http://localhost/ScrumTaskboard/login	20	225	10	3011	645.95	0.00%	1.7/min	0.90	32165.9
http://localhost/ScrumTaskboard/showDevelopers	20	293926	241155	324816	24910.12	0.00%	1.8/min	105.01	3615101.8
http://localhost/ScrumTaskboard/showMenu	30	697	12	6066	1430.07	0.00%	15.5/min	0.61	2416.3
http://localhost/ScrumTaskboard/showTaskboar...	10	23253	409	58141	13890.03	0.00%	5.8/min	22.91	244658.5
http://localhost/ScrumTaskboard/addTask	20	834	19	3272	1063.41	0.00%	11.2/min	2.83	15483.2
TOTAL	120	51275	10	324816	109321.75	0.00%	10.3/min	108.02	644786.1

Figure 4. Summary report – 10 users.

determine which HTTP request consumed most of the CPU's activities and propose corrective measures oriented towards reducing the load on the server and obtaining a better throughput value.

More precisely, by analyzing the output of the Summary Report listeners, we can easily determine the requests that represent possible sources of server failure or slow responsiveness, thus acting as real bottlenecks. As a maximum throughput is desired, high attention should be given to small values reported per minute time unit.

An essential point to mention is the interconnection among the throughput and the average response time listed in the same table under the Average column of the Summary Report listener. By analyzing this summarized report (Figure 4), one may notice an inverse relation between these two metrics, thus an increase in the throughput value determines a decrease of the response time required for the web server to handle the corresponding HTTP request. Alternatively, the Average response time column of the Summary Report might be taken as reference for identifying the system's bottlenecks denoted as considerable high values for one or several HTTP requests recorded in the use case scenario. Starting from these time consuming indicators, the associated throughput is analyzed and determined as small valued result reported per minute time unit.

Specifically for the case scenario under test (login into the application, visualize all developers, then visualize the taskboard, add a new, revisit the list of developers and finally the log out), executed by 10 concurrent worker threads, Figure 4 illustrates the significantly high response time for the /showDevelopers HTTP request (as compared to all the other ones recorded in this use case) associated with a considerable small throughput, perceived as problematic performance features of the system under test.

Proceeding with an increased number of 30 users that simultaneously execute the considered use case, the Summary Report listener (Figure 2) serves as source of performance analysis. Therefore, by exploring both Average and Throughput columns of the table formatted result, the same issues are identified for the /showDevelopers HTTP request, indicated by a significantly high response time corresponding to a minimum throughput. However, by comparing the results obtained from the Summary Report listener in the context of 10 and 30 concurrent threads (Figures 4 and 2), respectively, one may denote that the response time (reported under the Average column) is three times higher in last case (of 30 users) while the throughput is slightly decreased (from 10.3/min to 9.3/min in average).

Consequently, by measuring the throughput performance indicator in conjunction with the response time, the presented usability monitoring approach illustrates the web server's behavior. The conducted evaluations were performed in the context of varying the number of users concurrently accessing the application.

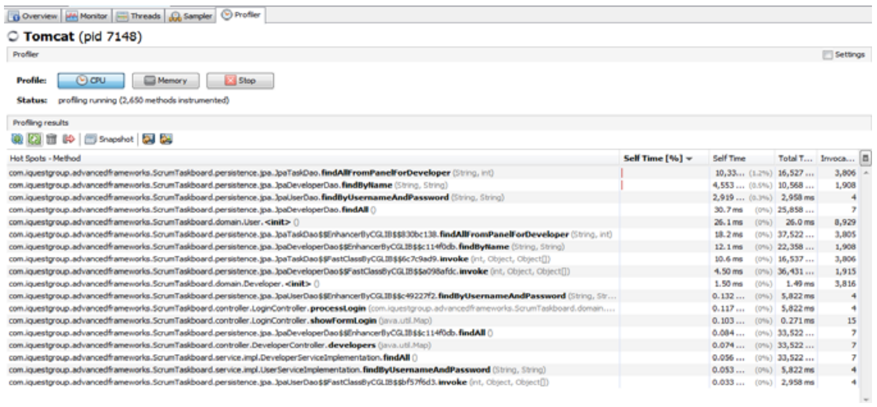
As bottom line of this subsection, we may conclude the problematic behavior of the web server concerning performance aspects might be determined by either evaluating the reported percentage error as well as the server's throughput and response time metrics. By following any of these approaches, the performance flaws of the CPU consuming requests were determined for a variable number of concurrent customers, thus denoting poor usability, scalability, response time and other performance measurements.

Step 2: Determine the root cause of the usability issues

The next step in the proposed methodology is to successfully determine the root cause of the encountered responsiveness-oriented performance issues so that to further on establish concrete solutions to be implemented in order to increase the usability and overall performance of the application. To accomplish this, the simulated test cases executed by taking advantage of the functionalities offered by JMeter, were associated with the ones provided by VisualVM performance monitoring tool (VisualVM, 2016). VisualVM offered performance results reported to different diagnosis type: CPU time and memory.

VisualVM as an application performance monitoring tool which freely comes when installing Java JDK, offers several monitoring options, among which the profiling and sampling ones are of interest for our research. The profiler and sampler tools accessible from the VisualVM's IDE are available for both CPU and memory load as performance indicators.

By instrumenting the web application under test, profiling adds a constant amount of extra execution time to every single method call. Sometimes, this results in adding large amount of time to the execution which may even last hours. VisualVM's sampler works by taking a dump of all of the threads of execution on a fairly regular basis, and uses this to determine how much CPU time each method spends. Usually, sampling takes a constant amount of time each second to record stack traces for each thread, thus only adding 5 – 10 minutes of execution time in total, while still succeeding to provide information regarding the source causes of potential performance vulnerabilities. However, sampling has the drawback that the



Tomcat (pid 7148)

Profiler

Profile: CPU Memory Stop

Status: profiling running (2,650 methods instrumented)

Profiling results

Hot Spots - Method	Self Time [%]	Self Time	Total T...	Invoca...
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaTaskDao.**findAllFromPanelForDeveloper** (String, int)		10,33... (1.2%)	16,527 ...	3,806
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaDeveloperDao.**findByName** (String, String)		4,553 ... (0.5%)	10,568 ...	1,908
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaUserDao.**findByUsernameAndPassword** (String, String)		2,919 ... (0.3%)	2,958 ms	4
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaDeveloperDao.**findAll** ()		30.7 ms (0%)	25,858 ...	7
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.User. **<init>** ()		26.1 ms (0%)	26.0 ms	8,929
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaTaskDao$$EnhancerByCGLIB$$830bc138.**findAllFromPanelForDeveloper** (String, int)		18.2 ms (0%)	37,522 ...	3,805
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaDeveloperDao$$EnhancerByCGLIB$$c114f0db.**findByName** (String, String)		12.1 ms (0%)	22,358 ...	1,908
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaTaskDao$$FastClassByCGLIB$$6c7c9ad9.**invoke** (int, Object, Object[])		10.6 ms (0%)	16,537 ...	3,806
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaDeveloperDao$$FastClassByCGLIB$$a098afdc.**invoke** (int, Object, Object[])		4.50 ms (0%)	36,431 ...	1,915
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.Developer. **<init>** ()		1.50 ms (0%)	1.49 ms	3,816
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaUserDao$$EnhancerByCGLIB$$c49227f2.**findByUsernameAndPassword** (String, Str...		0.132 ... (0%)	5,822 ms	4
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.LoginController.**processLogin** (com.iquestgroup.advancedframeworks.ScrumTaskboard.domain....		0.117 ... (0%)	5,822 ms	4
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.LoginController.**showFormLogin** (java.util.Map)		0.103 ... (0%)	0.271 ms	15
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaDeveloperDao$$EnhancerByCGLIB$$c114f0db.**findAll** ()		0.084 ... (0%)	33,522 ...	7
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.DeveloperController.**developers** (java.util.Map)		0.074 ... (0%)	33,522 ...	7
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.DeveloperServiceImplementation.**findAll** ()		0.056 ... (0%)	33,522 ...	7
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.UserServiceImplementation.**findByUsernameAndPassword** (String, String)		0.053 ... (0%)	5,822 ms	4
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa..JpaUserDao$$FastClassByCGLIB$$bf57f6d3.**invoke** (int, Object, Object[])		0.033 ... (0%)	2,958 ms	4

Figure 5. CPU usage information from VisualVM Profiler.

number of invocations recorded is not necessarily accurate, since a short method could easily start and finish between stack dumps. Therefore, the recommendation is to use sampling for recording the amount of CPU time or Memory consumption, rather than the number of invocations, thus having the chance to identify performance problems, much faster than the standard profiler.

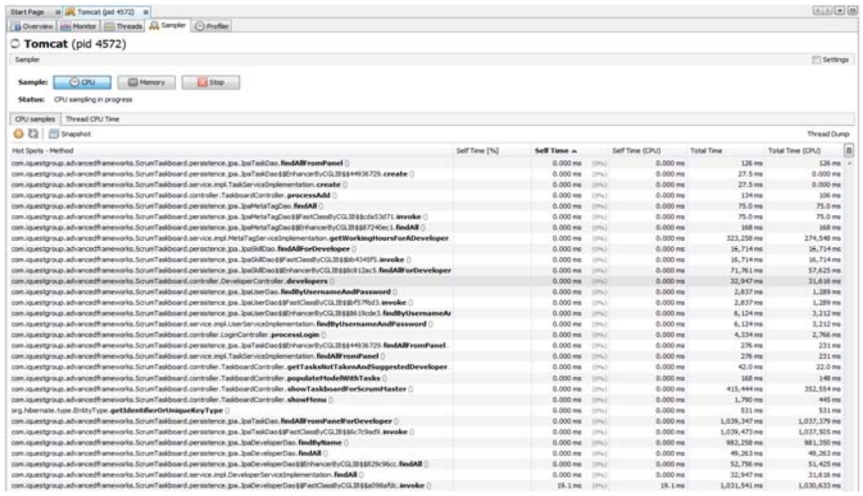
Therefore, both diagnosis-related results were analyzed and further on presented in a separate subsection as they represent two alternatives to be considered in this point of the proposed methodology. By selecting either of these two cause-oriented performance indicators and conducting a series of visual techniques, the system's bottlenecks might be identified and subsequently, specific solutions proposed so that to overcome the performance problems.

Consequently, determining the root cause of the performance issues faced by a web application, might be obtained by using as performance indicator the CPU load and the memory consumption.

CPU load as performance indicator

The VisualVM's profile command returns detailed data on method-level CPU performance, specifying the total execution time and number of invocations for each method. When analyzing application performance, VisualVM instruments all of the methods of the profiled application.

As a result, for the presented use case scenario (login into the application, visualize all developers, then visualize the taskboard, add a new task,



Tomcat (pid 4572)

Sampler

Sample:

Status: CPU sampling in progress

Hot Spots - Method	Self Time [%]	**Self Time**	Self Time (CPU)	Total Time	Total Time (CPU)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaTaskDao.**findAllFromPanel** ()		0.000 ms (0%)	0.000 ms	126 ms	126 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaTaskDao$$EnhancerByCGLIB$$44936729.**create** ()		0.000 ms (0%)	0.000 ms	27.5 ms	0.000 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.TaskServiceImplementation.**create** ()		0.000 ms (0%)	0.000 ms	27.5 ms	0.000 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.TaskboardController.**processAdd** ()		0.000 ms (0%)	0.000 ms	134 ms	106 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaMetaTagDao.**findAll** ()		0.000 ms (0%)	0.000 ms	75.0 ms	75.0 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaMetaTagDao$$FastClassByCGLIB$$cda53d71.**invoke** ()		0.000 ms (0%)	0.000 ms	75.0 ms	75.0 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaMetaTagDao$$EnhancerByCGLIB$$87240ec1.**findAll** ()		0.000 ms (0%)	0.000 ms	168 ms	168 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.MetaTagServiceImplementation.**getWorkingHoursForADeveloper** ...		0.000 ms (0%)	0.000 ms	323,258 ms	274,548 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaSkillDao.**findAllForDeveloper** ()		0.000 ms (0%)	0.000 ms	16,714 ms	16,714 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaSkillDao$$FastClassByCGLIB$$bb4345f5.**invoke** ()		0.000 ms (0%)	0.000 ms	16,714 ms	16,714 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaSkillDao$$EnhancerByCGLIB$$8c812ac5.**findAllForDeveloper**		0.000 ms (0%)	0.000 ms	71,761 ms	57,625 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.DeveloperController.**developers** ()		0.000 ms (0%)	0.000 ms	32,947 ms	31,616 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaUserDao.**findByUsernameAndPassword** ()		0.000 ms (0%)	0.000 ms	2,837 ms	1,289 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaUserDao$$FastClassByCGLIB$$bf57f6d3.**invoke** ()		0.000 ms (0%)	0.000 ms	2,837 ms	1,289 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaUserDao$$EnhancerByCGLIB$$8619cde3.**findByUsernameAr**		0.000 ms (0%)	0.000 ms	6,124 ms	3,212 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.UserServiceImplementation.**findByUsernameAndPassword** ()		0.000 ms (0%)	0.000 ms	6,124 ms	3,212 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.LoginController.**processLogin** ()		0.000 ms (0%)	0.000 ms	4,334 ms	2,766 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaTaskDao$$EnhancerByCGLIB$$44936729.**findAllFromPanel** ...		0.000 ms (0%)	0.000 ms	276 ms	231 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.TaskServiceImplementation.**findAllFromPanel** ()		0.000 ms (0%)	0.000 ms	276 ms	231 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.TaskboardController.**getTasksNotTakenAndSuggestedDeveloper** ...		0.000 ms (0%)	0.000 ms	42.0 ms	22.0 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.TaskboardController.**populateModelWithTasks** ()		0.000 ms (0%)	0.000 ms	168 ms	148 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.TaskboardController.**showTaskboardForScrumMaster** ()		0.000 ms (0%)	0.000 ms	415,444 ms	352,554 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.TaskboardController.**showMenu** ()		0.000 ms (0%)	0.000 ms	1,790 ms	445 ms
org.hibernate.type.EntityType.**getIdentifierOrUniqueKeyType** ()		0.000 ms (0%)	0.000 ms	531 ms	531 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaTaskDao.**findAllFromPanelForDeveloper** ()		0.000 ms (0%)	0.000 ms	1,039,347 ms	1,037,379 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaTaskDao$$FastClassByCGLIB$$6c7c9ad9.**invoke** ()		0.000 ms (0%)	0.000 ms	1,039,473 ms	1,037,505 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaDeveloperDao.**findByName** ()		0.000 ms (0%)	0.000 ms	982,258 ms	981,350 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaDeveloperDao.**findAll** ()		0.000 ms (0%)	0.000 ms	49,263 ms	49,263 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaDeveloperDao$$EnhancerByCGLIB$$829c96cc.**findAll** ()		0.000 ms (0%)	0.000 ms	52,756 ms	51,425 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.DeveloperServiceImplementation.**findAll** ()		0.000 ms (0%)	0.000 ms	32,947 ms	31,616 ms
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.JpaDeveloperDao$$FastClassByCGLIB$$a098afdc.**invoke** ()		19.1 ms (0%)	19.1 ms	1,031,541 ms	1,030,633 ms

Figure 6. CPU usage information from VisualVM Sampler.

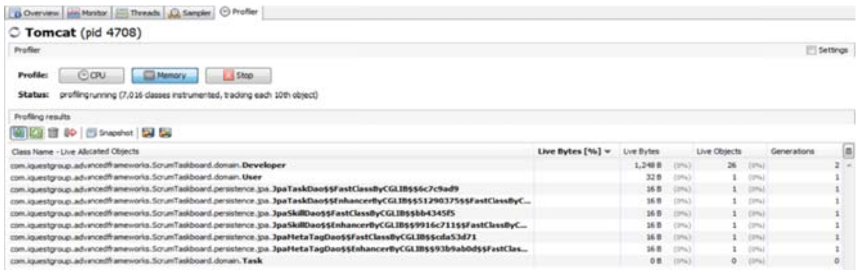
together with the required information for it, visit again the list of developers and finally the log out from the application) executed in the context of 10 concurrent threads (configured in JMeter), the main bottleneck(s) of the system were determined by using both profiler and sampler tools. The following two images present CPU usage information for profiling and sampling of the considered use case (Figure 5 and Figure 6).

These two images offer information related to the bottleneck points of the application, by listing the specific functionalities that consume most of CPU's time. By taking advantage of these results, corrective, problem-oriented solution might be implemented in order to overcome the performance vulnerabilities faced by the application.

Memory consumption as performance indicator

When analyzing memory usage, VisualVM's profiler starts instrumenting the loaded classes and displays the total number of objects allocated by each class (including array classes) in a tabled format. For each class currently loaded class in the Java Virtual Machine (JVM), the profiling results display the size and number of objects allocated since the profiling session started. The results are automatically updated as new objects are allocated and as new classes are loaded.

Similar to the previously presented approach to monitor the system's performance flaws by analyzing CPU load, the memory-oriented one was



Tomcat (pid 4708)

Profiler

Profile:

Status: profilingrunning (7,016 classes instrumented, tracking each 10th object)

Profiling results

Class Name - Live Allocated Objects	Live Bytes [%]	Live Bytes		Live Objects		Generations
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**Developer**		1,248 B	(0%)	26	(0%)	2
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**User**		32 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskDao$$FastClassByCGLIB$$6c7c9ad9**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskDao$$EnhancerByCGLIB$$51290375$$FastClassByC...**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaSkillDao$$FastClassByCGLIB$$bb4345f5**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaSkillDao$$EnhancerByCGLIB$$9916c711$$FastClassByC...**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaMetaTagDao$$FastClassByCGLIB$$cda53d71**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaMetaTagDao$$EnhancerByCGLIB$$93b9ab0d$$FastClas...**		16 B	(0%)	1	(0%)	1
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**Task**		0 B	(0%)	0	(0%)	0

Figure 7. Memory consumption information from VisualVM

undergone in the context of 10 users that simultaneously execute the sample use case scenario. The outcome obtained from profiling and sampling the considered scenario (login into the system, visualize all developers, visualize the taskboard, add a new task, revisit the list of developers and logout from the application) is illustrated in Figures 7 and 8.

These results provide exact information regarding the memory usage of the system under test, by indicating the amount of objects created as well as the class allocating them. This data is of great help for removing potential memory leak problems and enhancing the application's performance characteristics, by setting up theclean-up or garbage collection mechanisms.

Consequently, the association between the JMeter and VisualVM tools, provide accurate information regarding the precise source of failure for the considered web application. Furthermore, the results obtained are oriented towards both CPU and Memory usage, which enable specific solutions to be designed and implemented.

Step 3: Analyze results and propose solutions to be implemented in order to overcome the usability issues

By analyzing the results obtained from VisualVM, visual techniques might be used to determine the root cause of the usability issues. Moreover, as in the previous step of the proposed methodology the root cause of a web application's usability issue was determined using both CPU and Memory consumption as performance indicators, the results obtained from VisualVM's profiling and sampling tools are specialized for the selected indicator type (CPU or Memory consumption).

Consequently, the results interpretation as well as possible solutions proposal is presented in the following sections, separated by the performance diagnosis used.

Solutions proposed to solve the CPU-related usability issues

Starting from the results obtained by profiling and sampling activities (Figures 5 and 6) for the considered use case scenario (login into the system, visualize all developers, visualize the taskboard, add a new task, revisit the list of developers and log out from the application), it can be determined that most of the CPU time is spent on calling functionality at the persistence layer. This fact is proved by the series of database methods invocation that are reported in the VisualVM's profiler and sampler as being the most CPU consuming spots. More precisely, Figures 5 and 6 illustrate the method call hierarchy, ordered by their CPU time consumption when the considered use case scenario is performed:

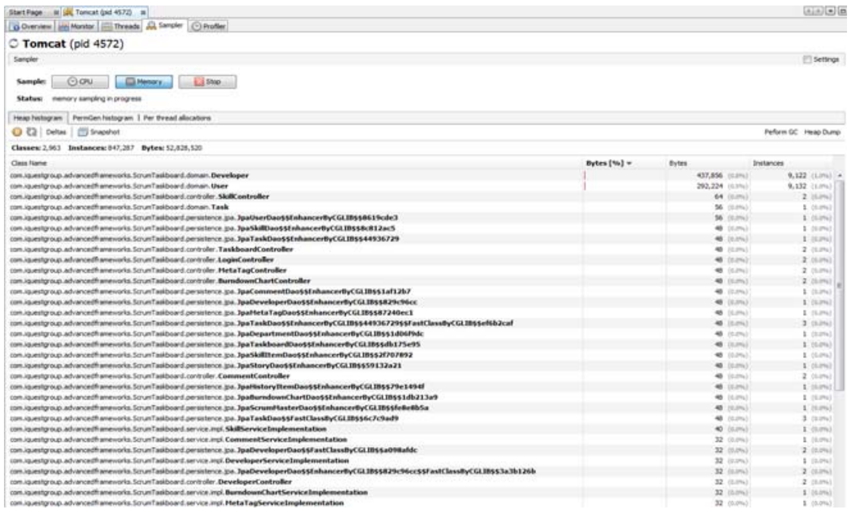
```
findAllFromPanelForDeveloper(),
findByName(),
findByUsernameAndPassword(),
findAll(),
findAllFromPanel(),
findAllForDeveloper().
```

Therefore, the main CPU bottleneck is caused by methods from the persistence layer (when querying the database) and thus, further improvements should be implemented at the database level.

More precisely, for removing the database bottleneck, there are several techniques that may be explored and appropriate solutions implemented. By studying documentation related to database enhancements and methods to be applied on a web application to improve the persistence layer access, a series of rules and guidelines were considered. As one of the main strategy to be applied at persistence layer is adding database indexes where appropriate, this topic was studied in detail and best practices taken into account (Garner, 2007).

According to Singh (1998), in order to fix the database bottleneck which determined the application to become unresponsive when a large number of users were trying to access it, the persistence layer strategies were applied where appropriate and the results analyzed after each step. Specifically, for the described use case scenario, by means of SQL syntax, slow queries were determined and database indexes implemented in the adequate manner.

As described by Shirazi (2003), another performance tuning strategy to remove slow database queries is to add method-level caching. There are several available methods for caching a certain functionality in Java, depending on the application specifics and needs. For instance, after



Tomcat (pid 4572)

Sampler

Sample:

Status: memory sampling in progress

Classes: 2,963 **Instances:** 847,287 **Bytes:** 52,828,520

Class Name	Bytes [%]	Bytes	Instances
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**Developer**		437,856 (0.8%)	9,122 (1.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**User**		292,224 (0.5%)	9,132 (1.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**SkillController**		64 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.domain.**Task**		56 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaUserDao$$EnhancerByCGLIB$$8619cde3**		56 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaSkillDao$$EnhancerByCGLIB$$8c812ac5**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskDao$$EnhancerByCGLIB$$44936729**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**TaskboardController**		48 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**LoginController**		48 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**MetaTagController**		48 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**BurndownChartController**		48 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaCommentDao$$EnhancerByCGLIB$$1af12b7**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaDeveloperDao$$EnhancerByCGLIB$$829c96cc**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaMetaTagDao$$EnhancerByCGLIB$$87240ec1**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskDao$$EnhancerByCGLIB$$44936729$$FastClassByCGLIB$$ef6b2caf**		48 (0.0%)	3 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaDepartmentDao$$EnhancerByCGLIB$$1d06f9dc**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskboardDao$$EnhancerByCGLIB$$db175e95**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaSkillItemDao$$EnhancerByCGLIB$$2f707892**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaStoryDao$$EnhancerByCGLIB$$59132a21**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**CommentController**		48 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaHistoryItemDao$$EnhancerByCGLIB$$79e1494f**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaBurndownChartDao$$EnhancerByCGLIB$$1db213a9**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaScrumMasterDao$$EnhancerByCGLIB$$fe8e8b5a**		48 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaTaskDao$$FastClassByCGLIB$$6c7c9ad9**		48 (0.0%)	3 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.**SkillServiceImplementation**		40 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.**CommentServiceImplementation**		32 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaDeveloperDao$$FastClassByCGLIB$$a098afdc**		32 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.**DeveloperServiceImplementation**		32 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.persistence.jpa.**JpaDeveloperDao$$EnhancerByCGLIB$$829c96cc$$FastClassByCGLIB$$3a3b126b**		32 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.controller.**DeveloperController**		32 (0.0%)	2 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.**BurndownChartServiceImplementation**		32 (0.0%)	1 (0.0%)
com.iquestgroup.advancedframeworks.ScrumTaskboard.service.impl.**MetaTagServiceImplementation**		32 (0.0%)	1 (0.0%)

Figure 8. Memory consumption information from VisualVM Sampler.

studying optimization oriented best practices and guidelines (Oransa, 2014) and taking into account the specifics of the web application under test, the recommendation is to develop method-level caching using Spring eh-cache. Being the most widely-used Java-based cache, eh-cache methodology is robust, proven, and full-featured.

Solutions proposed to solve the Memory-related usability issues

As far as Memory consumption concerns, VisualVM's profiler and sampler offer accurate information regarding the amount of objects created of a certain type when the sample use case is being executed. As a result, by interpreting the results from Figures 7 and 8, one may easily determine that the largest amount of instances populating the Memory are model classes used in the web application. More precisely, domain POJO (Plain Old Java Objects) classes such as Developer, User, Task, are reported to allocate most of the Memory' space.

In order to reduce the amount of such model objects creation, study and research activities were conducted and insights into this topic were gathered. According to Higgins (2007), one valid strategy to be considered when trying to overcome the Memory leak problems is to perform clean-up or garbage collection of the redundant objects instances populating the Memory.

Fisher and Murphy (2016) proposed entity-caching as a performance boost method when facing performance issues caused by large amount of entity objects. In this direction, Hibernate eh-caching might be easily used inside a Spring-based web application.

Another solution intended to solve the Memory consumption problem is to replace the usage of object model classes inside service methods as much as possible with DTO (Data Transport Object) ones. The reason behind this recommendation is to create lightweight DTO classes which will contain only the required attributes from the initial object model. More precisely, only those attributes of the model class used inside the persistence layer (for database queries) or service level one will populate the newly created DTO object that will be further on used in all business logic procedures throughout the code.

Finally, as a general solution to optimize both CPU and Memory consumption and therefore tune the application's performance is to implement changes at the Apache server level in order to remove the ProxyError reported by JMeter's View Results Tree listener (Figure 1) in the context of a large number of concurrent threads accessing the application. In order to develop such configurations, additional research and investigation have to be performed so that to obtain an insight of proper Apache parameters' value (timeout, keepalive, and so on).

Consequently, by means of the visual techniques experienced, the root causes of the encountered usability problems were briefly identified and, categorized based on diagnosis indicator (CPU and Memory consumption). Furthermore, taking into account the conducted research, technical solutions might be proposed and corresponding approached implemented, thus enhancing the overall system's performance.

Step 4: Evaluating the implemented solutions

When assessing the performance enhancements brought by the developed solutions, considerations have to be related to the diagnosis indicators. Therefore, the evaluation phase is conducted on two separate subsections regarding the CPU load and the Memory consumption.

CPU load as performance indicator

By implementing the proposed solution for the reported CPU load vulnerabilities, the encountered bottleneck at database level was completely removed for the considered use case and the initial error percentages significantly reduced. The developed technical approach was based on an algorithm to determine the database queries which are reported as slow and evaluate the usefulness of adding an index. More precisely, after adding an index on a database table's column, the initial reported error was half reduced. Further on, several analyses were conducted in order to determine the suitability of database indexing for the problematic SQL queries, and indexes added where appropriate. This strategy brought successive performance improvement and the usability, and scalability, issues were reduced, fact denoted by the minimized reported error percentage for a large number of concurrent users.

Also, by implementing method-level caching for the long-running database searches (denoted by the many findX() methods from VisualVM reports) using Spring eh-cache approach, further performance enhancements are obtained. For reducing the CPU load, cache is developed on the service-level functionality which invokes the persistence layer methods reported as problematic. However, caching has its drawbacks and therefore, should be carefully used and performance metrics measured.

Memory consumption as performance indicator

By modeling Hibernate eh-caching inside a Spring-based web application, Memory consuming entity classes might be cached, thus speeding up the process. Moreover, adding an intermediary layer in the application for DTOs (Data Transport Objects) will definitely tune the system's performance as the heavy loaded entity classes are replaced by their lightweight clones.

As far as Memory consumption regards, there are clean-up and garbage collection activities that should be investigated and considered as valuable solutions to deallocate unused objects from the Memory.

3.4 Validation of the implemented methodology

Therefore, by using the aforementioned visual strategies by following the described methodology steps, the root cause of the usability issue was determined and technical solutions were considered and successfully applied. The overall result was an increase in the user experience, satisfaction and response time metrics throughout the system. Consequently, the application will manage not to slow, block or frustrate concurrent customer requests' fulfillment, which determine client loss.

Furthermore, apart from the database bottleneck and the Memory allocation with large amount of object model instances obtained for CPU and Memory load as performance indicators, the source code was revised and analyzed. More precisely, some refactoring strategies were proposed for the functionalities reported as causing performance vulnerabilities. Consequently, the next step in the usability enhancement process was the development of these strategies or even re-implementing of the actual solution (where appropriate) so that to remove possible time or memory consuming algorithms and replace them with a cleaner, cost-effective approach. By applying performance improvement methods to other use case scenarios (apart from the one exemplified in this paper) the system's behavior was definitely enhanced.

To conclude this chapter, the proposed methodology was of great help to determine the existence of usability issues, identify their root causes and allow proposal of particular technical solutions to overcome them. This way, the main operations were performed at the very bottleneck points of the system and immediate recovery occurred. As a result, this boosts the overall system performance and offered the desired user experience because of improved usability.

4. Conclusions and Further Work

As briefly presented in the previous chapter, by following the proposed methodology, the web application's performance problems, oriented towards serviceability, operability and user responsiveness, were spotted and the context clearly defined. Furthermore, by using tools such as JMeter and VisualVM, the obtained results were visually analyzed and the root cause of the issue determined. This way, particular strategies were proposed, centered on fixing the bottleneck, rather than offering a general system improvement (which would only slightly enhance the performance, the root cause not being handled yet).

This methodology proved successful for the considered context and it aimed to enhance the overall system's performance by focusing on the problems identified, i.e. its response time capabilities when increasing the usage workload (defined as the number of concurrent active users). However, attention must be paid to the specifics of a certain system whose performance is to be evaluated and further on be improved, because they may differ and therefore, several adjustments have to be applied to the proposed methodology in order to fit particular needs.

Therefore, by applying the described methodologies to fix the encountered problem, system's usability was enhanced, fact clearly proven by the conducted experiments. Not only did the conducted results denote the system's improved performance-centered features by the considerably reduced reported error and enhanced throughput, but also the methodology validation indicates its efficiency. More precisely, the obvious advantages and strengths of the presented methodology are related to speed in analyzing the system's non-functional requirements from the customer's experience and serviceability viewpoint, as well as accuracy in determining the main bottleneck. The specified speed capabilities refer to the automatisation of the testing process by using the JMeter tool which allows to simulate the number of concurrent users, otherwise being almost impossible to assess the system's behavior in this context (as manual testing would be more than cumbersome). As far as the accuracy concerns, the presented methodology offers a precise diagnosis of the encountered response time-based user operability problems by identifying their root cause. This way, specific problem-oriented solutions might be designed and implemented. By focusing on the source of failure, corrective measures are taken at an early stage in the development process.

However, there are still several strategies to be implemented that would obviously boost the entire application performance, and offer a better user experience. More precisely, a technical solution such as setting up a load balancer represents only one example that involves further study, analysis and implementation effort. All these potential strategies have to be tested to determine their suitability for our particular web application and establish practical solutions to be developed in order to successfully apply them.

References

Barish, G., Building Scalable and High-Performance Java Web Applications Using J2EE Technology, Addison-Wesley Professional, 2002.

Fisher, P., Murphy, B.D., *Spring Persistence with Hibernate*, 2016.

Garner, S. R., Data Warehouse Implementation Strategies: A Mixed Method Analysis of Critical Success Factors, ProQuest, 2007.

Haralambos, M., R., Colette, Advanced Information System Engineering, *23rd International Conference, CAiSE 2011*, London, 2011, Proceedings.

Higgins, K. R., An Evaluation of the Performance and Database Access Strategies of Java Object-Relational Mapping Frameworks, ProQuest, 2007.

JMeter tool, Online documentation (2016), http://jmeter.apache.org/

Mendes, E., Mosley,N., *Web Engineering*, Springer Science and Business Media, 2006.

Mohler, J. L., Duff, J. M., *Designing Interactive Web Sites*, Delmar/Thomson Learning, 2000.

Olsina, L.; Rossi, G.; 1999, Towards Web-site Quantitative Evaluation: defining Quality Characteristics and Attributes, Proceedings of *IV Int'l WebNet Conference*, World Conference on the WWW and Internet, Hawaü, US, Vol. 1, pp. 834-839.

Oransa, O., Java EE 7 Performance Tuning and Optimization, Packt Publishing Ltd, 2014.

Purba, S., Architectures for E-Business Systems: Building the Foundation for Tomorrow's Success, CRC Press, 2001.

Rossi, G. et al, *Web Engineering: Modelling and Implementing Web Applications*, Springer Science and Business Media, 2007.

Rubin, K.S., Essential Scrum: A Practical Guide to the Most Popular Agile Process, Michigan, USA, Pearson Education, 2013.

Schwaber, K. and Sutherland, J., The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game, July, 2013.

Shivakumar, S. K., Architecting High Performing, Scalable and Available Enterprise Web Applications, Morgan Kaufmann, 2014.

Singh, H., Data Warehousing: Concepts, Technologies, Implementations, and Management, Prentice Hall PTR, 1998.

Singh, K. K., A Quantitative Method for Evaluation of Websites Quality using WebQEM Tool, *Journal of Global Research Computer Science and Technology* (JGRCST), (2016), http://globalresearch.co.in/Global_Research_Vol_I_Iss_I_1.pdf

Stoneham, R., Dastbaz, M, *Building Interactive Web Applications*, Addison-Wesley Longman, Limited, 2006.

Shirazi, J., *Java Performance Tuning*, O’Reilly Media Inc., 2003.

VisualVM tool, Online documentation (2016), https://visualvm.java.net/