بسم الله الرحمن الرحيم

Sudan University of Science &Technology

College of Graduate Studies

College of Computer Science and Technology

Evaluation of Code Quality in Web Pages using Software Testing Tools

تقويم جودة البرنامج في صفحات الويب باستخدام أدوات اختبار البرامج

Prepared by:

Maaza Eldaleel Mustafa Eldaleel

Supervisor

Dr. Mohamed El-Ghazali Hamza Khalil

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree Master of Computer Science

November 2019

Declaration

I hereby declare that the work reported in this M.Sc. thesis titled as "An Evaluation of Test Coverage Code Quality Tools and Response Analysis in Web Pages" submitted at the Master of Science degree in Computer Science, is an authentic record of my work carried out under the supervision Dr. Mohammed Elghazali Hamza. I have not submitted this work elsewhere for any other degree.

Maaza Eldaleel Mustafa Eldaleel	Dr Mohammed Al-Ghazali Hamza
---------------------	Supervisor

Acknowledgement

I would first like to thank my thesis advisor Dr. Mohammed Elghazali Hamza Khalil of the Faculty of Computer Science at Sudan University of Science and Technology. The door to Dr. Elghazali office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this research to be my own work, but steered me in the right direction whenever he thought I needed it.

Abstract

Continuous improvement is a key factor for survival in today's turbulent business environment. Correspondingly, in today's business world, software is everywhere. In the whole process of software development, testing is a phase that is often forgotten. Everybody assumes that once the software is developed, it will work flawlessly. However, it often happens not to be so. Moreover, when it does not, we are all unsatisfied and frustrated. Problems with software are a frequent occurrence, but only a few people address them in their product. The problems caused by a bad code quality vary from bugs to project delays and in general cause issues with the project timeline and budget. Furthermore, software testing has come to play a vital role in the success of the business, it makes sure that the application's performances are adequate and the customers are satisfied with it. When the delivered product is of quality, it helps in gaining the confidence of the customers. Through a load testing using webload and apache JMeter tools on a number of websites such as(Apple.com) and site from own design (pstore.com), interactive social site (noonpost.com), number of performance measurements based on the analysis results of the two tools were evaluated. This research presented also automated unit testing to provide dependable, periodic feedback to project developers. Constant feedback allows them to correct code as part of the development process and it reduces the checking time.

المستخلص

التحسين المستمرعامل رئيسي للبقاء في بيئة الأعمال المضطربة اليوم. بالمثل ، أصبح الآن في عالم الأعمال اليوم ، البرامج في كل مكان. في عملية تطوير البرمجيات بأكملها ، يعد الاختبار مرحلة غالباً ما يتم نسيانها. الجميع يفترض أنه بمجرد تطوير البرنامج ، سوف يعمل بدون عيوب. ومع ذلك ، يحدث في كثير من الأحيان أخطاء. علاوة على ذلك ، فإننا جميعًا نكون غير راضين ومحبطين. تعد مشكلات البرامج أمرًا متكررًا ، لكن عددًا قليلاً فقط من الأشخاص يعالجونها في برامجهم.

تختلف المشكلات الناتجة عن جودة الكود السيء من الأخطاء إلى تأخيرات المشروع وفي القضايا العامة مشاكل تتعلق بالجدول الزمني للمشروع والميزانية. علاوة على ذلك ، أصبح اختبار البرمجيات يلعب دورًا حيويًا في نجاح الأعمال ، فهو يضمن أن أداء التطبيق كافٍ وأن العملاء راضون عنه. عندما يكون المنتج الذي تم تسليمه عالي الجودة ، فإنه يساعد في اكتساب ثقة العملاء.

من خلال اختبار الحمل باستخدام أدوات webload و apache JMeter على عدد من مواقع الويب مثل(Apple.com)والموقع (pstore.com) من تصميمي، والموقع الاجتماعي التفاعلي (noonpost.com)، عدد من قياسات الأداء استنادًا إلى نتائج تحليل الاداتين تم تقييمها.

قدم هذا البحث أيضًا اختبار الوحدة الآلي لتوفير تعليقات دورية، يمكن الاعتماد عليها لمطوري المشروع. تتيح لهم الملاحظات المستمرة تصحيح التعليمات البرمجية كجزء من عملية التطوير وتقليل وقت الاختبار.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

CI	Continuous Integration.
IDE	Integrated Development Environment
ROI	Return on Investment.
SDLC	Systems Development Life Cycle
STLC	Software Testing Life Cycle.
WA	Web Applications
QA	Quality assurance
ISP	Internet service provider
DB	Database
AWS	Amazon Web Services
SUT	System Under Test

Chapter ONE

INTRODUCTION

Chapter One

Introduction

Overview

In the last years, web applications (WA) have become one of the very common ways people interact with computers thanks to the increasing internet speed. WA is capable of more complex operations and provides various functionalities. In many instances, they tend to replace classic desktop applications for many reasons, e.g. cross-platform compatibility support or ability to be easily updated and maintained.

However, while providing many advantages, there are specific weaknesses of which one has to be aware when implementing such an application. The main disadvantage stems from the fact that developer does not know what the hardware, software and network the application will run on. For example, a general drawback of WA is their slower response to user actions when compared to a reaction speed of desktop applications. It is often caused by inappropriate front-end development.

The levels of quality, scalability, and maintainability of software programs can be improved and measured through the utilization of functional and non-functional software testing tools throughout the software development process. One of the most important things on every project is keeping codebase consistent and ensure its quality over time, or, at least, build a road to achieve this goal in the future. Part of this job can be achieved automatically. In web-land, it seems more common to test back-end code than front-end this is probably changing as tooling gets better.

Quality assurance can be done by manually operating and manipulating the SUT in different ways, using different inputs and comparing the actual outcome with the expectations and making sure that they align. Manual testing is by its very nature time-consuming and prone to errors. If testing is carried out in this way, for example by the same software developer who wrote the software, there’s also a risk that the test cases may be chosen based on how the code is expected to work. In the worst-case scenario, if a formal testing protocol is not defined or properly followed, there is also a chance of parts or all the quality assurance process being neglected [1].

Performance is a huge subject, but it's not always a "back-end" or an "admin" subject: it's a Front-End responsibility too. The Front-End Performance Checklist is an exhaustive list of elements check or at least be aware of, as a Front-End developer and apply to your project when (personal or professional) [2].

Web applications are a mixture of server-side and client-side code. Application can have performance problems on either side or both needs to be optimized. The client-side relates to performance as seen within the web browser. This includes initial page load time, downloading all of the resources, JavaScript that runs in the browser. Web performance optimization occurs by monitoring and analyzing the performance of your web application and identifying ways to improve it [3].

The motivation to enforce good code is to quality improve the image of the company and help to create long-lasting partnerships with clients. Discussed tools in this research, that help developers and companies deploy software with good quality code by automatically perform checks on different areas of front end development.

Producing high-quality code is an aim that almost everyone in software development would say they support. Yet, long-term observations in the field reveal that many organizations do not back up thatworthy sentiment with the necessary resources (technology, budget,time, training, and management attention) or with the institutionalprocesses required to ensure that the quality of code is routinely maintained[4].

Software bugs have been around for as long as the software itself. Even for a program that has been thoroughly tested, the bugs are inevitable. The longer they are present in the code, the more expensive and difficult it is for them to be removed to boost front-end performance as much as possible for our developer. It is not so easy to convince every client to follow all of our performance guidelines. Correspondingly, to convince them by talking to them in their language, and explain the importance of performance for conversion or compare their performance to their main competitors. Our goal was to take control, focus on performance, be flexible for the future and to suggest recommendations for how to design, implement and administer tools for code quality in large-scale front-end software development projects.

1.1 Background of Research

Every year, billions of dollars are lost due to software that does not perform as expected. While the conformance to the customer requirements is mostly validated by functional testing, there is more to software quality than that. To cover the attributes like maintenance, readability, reusability or testability, that directly affect customers' perception of the product's quality, other inspection techniques need to be involved as well. There are different possibilities ranging from formal inspections of the code, peer code reviews, pair programming, to use of automated tools for static code analysis, which can be applied to ensure the quality of the code. The key to achieving quality software lies in the combination of these techniques [5].

According to this definition, quality assurance can simply be interpreted as assuring customer satisfaction [6]. According to IEEE Standard 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology" [7] “testing” is defined as: "The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component."

Performance is a vital factor deciding the quality of a web application, most of the performance issues happen on the front-end because developers pay much more attentions on the back-end dealing with business logic and data interaction. Front-End checklist items are created to go over a web application to check and boost the performance before releasing it to customers [8].

Even bad code can function. but if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. but it doesn't have to be that way. Ensuring code quality when your software team is growing rapidly is a huge challenge. But even with a constant number of software developers, maintaining code quality can cause headaches [9]. Without tools and a consistent system, the whole project can accumulate a huge technical debt, causing more problems in the long-term than it solves in the short-term.

1.2 Problem Statement

The problems caused by a bad code quality vary from bugs to project delays and in general cause issues with the project timeline and budget.

Low quality code can cause serious problems in the long term, which affect software quality attributes such as maintainability, performance and security of software systems.

1.3 Research questions

1. How to select a specific tool to maximize the code quality?

2. How to measure the performance in web pages through using the tools?

1.4 Objectives

- To determine specific tools to choose for maximizing code quality in software development
- To select guideline or framework for firms for excellent tools that are available and explanation and shows effectively in results.

1.5 Scope of Research

The scope of the research includes only front-end software development related coding issues and tools.

1.6 Proposed solution

Assessments of tools through measurement performance for web pages to improve code quality in development projects, and to suggest recommendations for how to design, implement and administer tools for code quality in software development projects.

1.7 Outlines the research

This thesis is divided into five chapters. Chapter two discusses a group of different attributes and requirements for code quality, related work. Chapter three about the research methodology, inside this chapter research design and procedures, were presented as well as tools, Chapter four which contain result and discussion about the implementation of work and finally the conclusion and future work at Chapter five.

Chapter Two

Literature Review

Chapter Two

Literature Review

2.1 Introduction

This chapter focuses on theoretical base information for the study such as code quality and front end. The first section briefly summarizes the concepts of software testing methodologies and our research based on briefly explains what is the web application is, the parts and the most important techniques used for development of code quality.

2.2 Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.

Continuous integration (CI) is a software engineering practice which reduces the blind spots of software development and leads to the process of building and delivering the software in rapid. At first, a daily build existed as standard. At present, each team member has to submit their work on daily basis and intended for a build to be accompanied with each significant alteration. When CI used by the book, provides constant feedback on the status of the product development. For the reason that CI discovers short ages initially during development process, defects are normally less significant, less complex and stress-free to resolve [10].

In terms of software quality, CI can help to measure cyclomatic complexity, code duplication, dependencies and coding standards so that developers can proactively refactor code before a defect is introduced. If a defect is introduced into a code base, CI can provide feedback soon after, when defects are less complex and less expensive to fix. Also, when using an effective developer testing regimen, CI provides quick feedback, via regression tests, on software that was previously working and is adversely affected by a new change.

Furthermore, to calculate how much time your team spends on fixing defects once the software is with the testing team, quality assurance, integration testing, etc. Of course, it costs considerably more once it enters production. When implementing effective CI practices, can discover many of these defects as soon as they are introduced, which significantly decreases the cost of fixing each defect.

The purpose of CI is to ensure required quality and consistency of the code in the development repository. Usually every code increment submitted by a developer is built with the newest code base and automatically checked against predefined criteria, like static code analysis and automated testing [11].

2.3 Attributes of Good Code Quality

Code quality is a combination of various attributes and conditions, which depend on the used business case. The following attributes are some of the most common ones for defining code quality, but there are tens of more attributes that can be used in addition depending on the code that is being evaluated:

- Clarity: Easy to read and oversee for anyone who isn't the creator of the code. If it's easy to understand, it's much easier to maintain and extend the code. Not just computers, but also humans need to understand it.
- Maintainable: A high-quality code isn't overcomplicated. Anyone working with the code has to understand the whole context of the code if they want to make any changes.
- Documented: The best thing is when the code is self-explaining, but it's always recommended to add comments to the code to explain its role and functions. It makes it much easier for anyone who didn't take part in writing the code to understand and maintain it.
- Refactored: Code formatting needs to be consistent and follow the language's coding conventions. Some code refactoring tips here.
- Well-tested: The fewer bugs the code has the higher its quality. Thorough testing filters out critical bugs ensuring that the software works the way it's intended.
- Extendible: The code received had to be extendible. It's not really great when to have to throw it away after a few weeks.
- Efficiency: High-quality code doesn't use unnecessary resources to perform the desired action.
- Tested: A code should have thorough tests to prevent any unwanted bugs from being introduced. Having well-written tests against the code will also enforce

maintainability and extensibility as the programmers can modify the code with ease of mind knowing that the tests will more likely show if something gets broken. Testing can be performed at many different levels in the software development process. Each level has been designed to detect different types of issues.

- Reliable: A code can be considered reliable when it works as expected and does not fail even in edge case situations. Reliable applications have fewer bugs and downtime and are therefore better for the business.
- Follows Standards: The code should follow any rules and regulations set by the client organization. The other programmers will understand easier the code and its maintenance takes less effort when it is following common standards. Following coding standards creates more consistent and uniform code regardless of each programmer’s styles.
- Reusability: Programming reuse is a decent cost productive and efficient advancement way. Distinctive code libraries classes ought to be sufficiently nonexclusive to utilize effectively in diverse application modules. Isolating application into diverse modules with the goal that modules can be reused over the application [12].

2.4 Dynamic Testing

Dynamic Testing is a kind of software testing technique using which the dynamic behaviour of the code is analyzed. For Performing dynamic, testing the software should be compiled and executed and parameters such as memory usage, CPU usage, response time and overall performance of the software are analyzed [13], is usually thought to have four different levels: unit testing, integration testing, system testing and acceptance testing.

Dynamic testing involves validation and it usually asks or checks "Are you building the right thing?” The software should be compiled and executed and input values are given and output values are checked with the expected output [14].

2.5 Unit Testing

Unit testing is defined as a type of software testing where individual units/ components of software are tested. Unit testing of software applications is done during the development (coding) of an application. The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming, a unit may be an individual function or procedure. Unit testing is usually performed by the developer.

In SDLC, STLC, V Model, Unit testing is the first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing. Dynamic testing involves testing the software for the input values, and output values are analyzed. Dynamic testing is the Validation part of Verification and Validation.

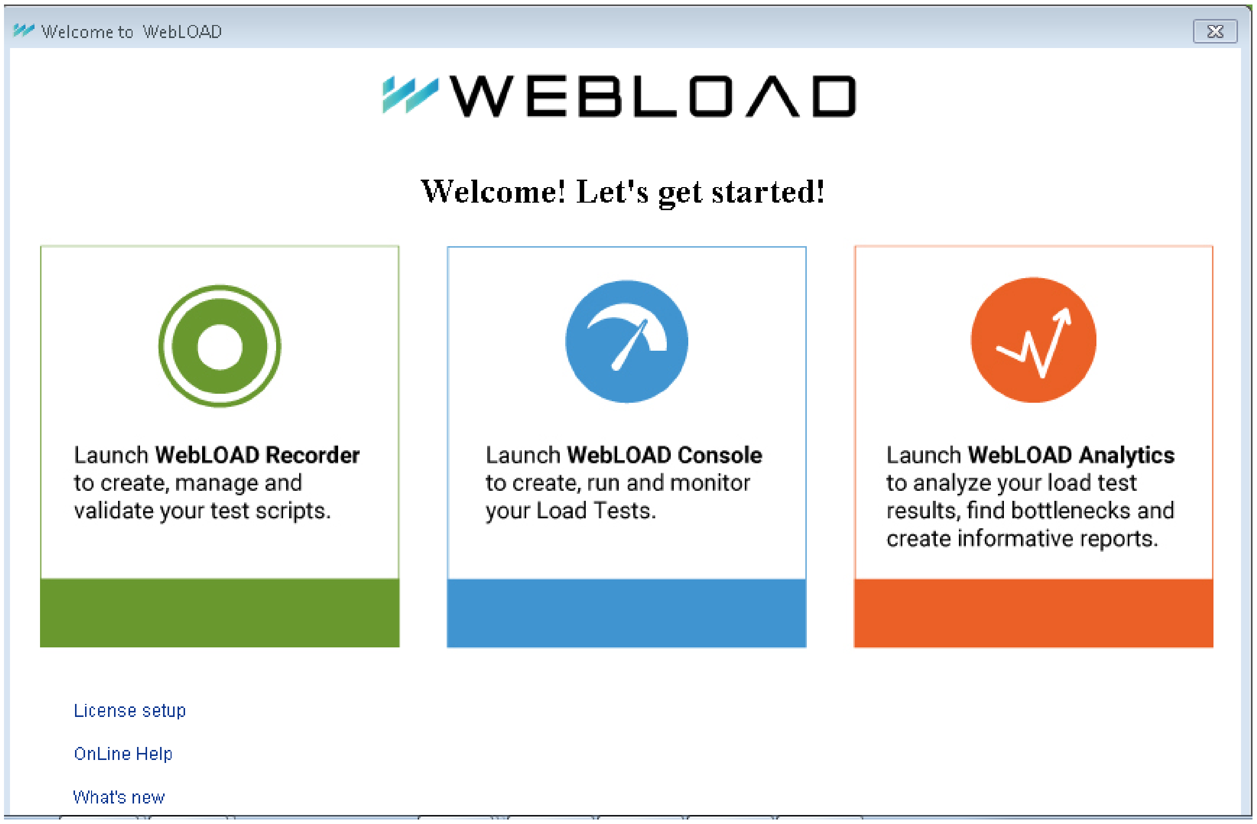
It is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software. It is also called module or component testing. It refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors [15].

2.6 WebLOAD Professional

WebLOAD is a website and performance testing tool for performing enterprise-scale load testing. Through AWS or cloud providers, can generate massive virtual user load through its Load Generation Console on Windows and Linux devices both in the cloud or locally. What separates it from other solutions is it's Free Edition for 50 virtual users that include community support and all of its features. Developed by RadView Software, WebLOAD is capable of supporting hundreds of technologies and features flexible built-in scripting capabilities.

Correspondingly, the side from offering extensive features for load testing web systems, WebLOAD is also capable of testing a variety of enterprise systems, including Oracle, SAP, and others. With WebLOAD, users can easily create and view their load script coupled with record and playback options, making test creation simple and efficient. The system implements multi-protocol support, enabling WebLOAD to replicate load

environments that number from hundreds to thousands of users and then provide accurate and detailed analysis of how the application behaves under load. With WebLOAD, businesses have a platform that allows them to perform load testing that is reliable, accurate, and efficient. The system offers an integrated development environment (IDE) that comes with a throng of useful features including correlation,
The system comes with a set of predefined analysis reports that provides developers with a detailed breakdown of performance data. This allows users to determine the bottlenecks and quickly make adjustments to resolve issues. The reports can also be viewed off-site through the customizable web dashboard.
WebLOAD also gathers critical information such as server-side statistics and other data from test runs. This helps users collect more information to apply within their root cause analysis. The platform is optimized for any device, ensuring that users can easily access the system from the desktop, laptop, and mobile phone, and analyze performance test results even if they are away from the office[34].



Welcome! Let's get started!

Launch **WebLOAD Recorder** to create, manage and validate your test scripts.

Launch **WebLOAD Console** to create, run and monitor your Load Tests.

Launch **WebLOAD Analytics** to analyze your load test results, find bottlenecks and create informative reports.

Figure 2.1: Welcome to Webload

2.7 CI test

Continuous Integration (CI) has evolved to become an integral part of the software ecosystem. When the discussion pivots to CI, the first tool that comes to mind is Jenkins. Jenkins is the most popular of any of the CI tools used today by software teams and is playing a significant role in the accelerating the 'Dev' in DevOps. It is an open source CI platform built in Java and has thousands of native integrations with useful tools. Jenkins enables developers to build, deploy, and automate projects; thereby, improving time to market and product quality.

As automation testing continues to help organizations scale their software quality efforts, it is imperative that automated tests are also part of the CI pipeline to achieve true continuous delivery. Most test types - including unit, integration, functional, and regression tests - are run via CI depending on the size of the test suite and the type of application under test.

Advantages of Jenkins include:

- It is an open source tool with great community support.
- It is easy to install. and share with the community.
- It is free of cost.
- It has 1000+ plugins to ease work. If a plugin does not exist can code it
- It is built with Java, and hence, it is portable to all the major platforms [35].

2.8 Related Work

Many research articles are found regarding identify ways to improve and enforce code quality using automated tools.

In [16] defining the quality of software-defined infrastructures. The outcome of the interviews suggests that among other important quality characteristics, testing is a very important quality characteristic.

On the other hand, [17] say in their research unit testing By definition and in practice, unit testing is done by the developer, not an independent tester or quality assurance person. Unit testing is based upon a structural view into the implementation code. Code coverage tools can be used to provide the programmer with insight into which part of the code structure has been exercised by tests. However, developers often do not use code coverage information in their determination that they have tested enough.

In [18] developed a framework and provide improvement suggestions for better quality optimization of current web page and further development process for same type of application.

The research in [19] propose two case study analyses conducted on past large-scale software development projects. the projects included some tools to improve the quality of the code, but they both also failed in some quality areas. These analyses were utilized to first find out the common pain points in software development projects, which reduced the code quality and secondly to help to identify methods that could solve these issues. Could be solved by enforcing the code quality with automated tools, caused many of the problems in the past projects. The set up of the projects determined a lot of the type of issues that arose, but most of them could have been avoided with a better planning and implementation of the quality enforcement tools. The results and suggestions consist of a list of tools and guidelines for how to use them to solve the most common coding related challenges.

In [20] in contrast, Avritzer and Weyuker introduced a new way to design an application-independent workload for doing such a performance evaluation. It was determined that for the given project, the quantity of software and system availability requirements made it impossible to port the system to the new platform in order to do the performance testing. Therefore, a novel approach was designed in which no software was ported at all. Instead, a synthetic workload was devised by tuning commercially-available benchmarks to approximate resource usage of the actual system.

The works in [21] discuss a case study describing the experience of using these approaches for testing the performance of a system used as a gateway in a large industrial client/server transaction processing application is presented.

According to [22], if test execution is performed manually then it is considered as inefficient and error prone whereas test execution which is performed automatically increases the efficiency in such a way it reduces the work of the testers. Automated test case execution helps in reducing the cost because it decreases human involvement.

In [23] the aim of web performance testing is to evaluate performances of the web application and all the back-end systems (DB and application server) changing the load. Performance testing activity guarantees system performances according to a defined load, identifying where response time is too high. Automated tools are used to generate and emulate the load of the end users. These kinds of tests are very useful to verify, in test plant, new releases of the application or to stress it with a greater load (useful, for example, before the launch of a new advertising campaign). An artificial traffic is generated to reproduce the activity of a certain number of users of different types and, if long response time are measured, to find out all the system bottlenecks. After a first tuning activity, all the tests should be executed again.

All the test activities can be executed on-site or in a remote way. The focus of the first kind of activity is the application (all the network involved between server web and browser is not considered) and it’s possible to extract lot of useful and detailed information (especially if, during the test, all the resources of the systems involved are under monitoring). With a remote test, as all the elements involved are considered (user browser, ISP, network and application) it’s possible to obtain an end-to-end measure of performance, but it’s impossible to get lot of details on every element.

The study in [24] introduces a model that assesses test code quality by combining source code metrics that reflect three main aspects of test code quality: completeness, effectiveness and maintainability. The model is inspired by the software quality model of the software improvement group which aggregates source code metrics into quality ratings based on benchmarking. To validate the model we assess the relation between test code quality, as measured by the model, and issue handling performance. An experiment is conducted in which the test code quality model is applied to 18 open source systems. The test quality ratings are tested for correlation with issue handling indicators, which are obtained by mining issue repositories. In particular, we study the (1) defect resolution speed, (2) throughput and (3) productivity issue handling metrics. The results reveal a significant positive correlation between test code quality and two out of the three issue handling metrics (throughput and productivity), indicating that good test code quality positively influences issue handling performance.

In [25] discussed comparative analysis of various software performance testing tools and their limitations and a new approach for software performance testing. With an upbeat approach, we can produce software that can meet performance objectives and can be delivered on time and within budget, avoiding the project crises.

In [26] provided a good insights of software testing practices Most of the teams (6(8), 75%) teams mentioned that their impression on the tool that they were using is satisfactory, Another team mentioned need of more tools to perform unit and integration testing, and also improvement in software testing repository.

In [27] they mentioned software testing types there are various software testing techniques as per the research and study like a black box, white box, grey box, regression, reliability, usability, performance, unit, system, integration, security, smoke, sanity and object-oriented testing etc. It is impossible to perform all types of testing on a software as there is always fixed amount of time allocated for testing. Functional testing is very common, and lots of research is done on them in the past that's why only in rare cases a site crashes due to lack of functional testing. The most recent failures that happened in the past are due to lack of Performance and Security testing. In 2014 Indian Railway site got crashed as it was not able to handle the load of customers. Another failure in 2014 is of Delhi University (DU) online application form web site crash on last day of submission due to excessive load on site. Then there were instances in 2013 when Indian government sites were hacked by some external agencies. After analyzing and survey of all these techniques, it is found that the right mix of testing types should be performed on a given software to ensure quality and overall reliable software.

This research focuses on the main testing techniques like functional [F], performance [P] and security testing[S]. The right mix of testing should be included from all headers of F, P and S. Functionality is the first and foremost aspect of software testing which ensures the quality of software. verification and validation are done using static and dynamic testing respectively. static testing involves all types of reviews, inspections, and walkthroughs. dynamic testing or actual validation involves all functional and non-functional testing types.

Recommended [28] of a pilot case study aiming: (a) to understand the implications of structural quality; and (b) to figure out the benefits of structural quality analysis of the

code delivered by open source style development. To this end, we have measured quality characteristics of 100 applications written for Linux, using a software measurement tool, and compared the results with the industrial standard that is proposed by the tool. Another target of this case study was to investigate the issue of modularity in open source as this characteristic is being considered crucial by the proponents of open source for this type of software development. we have empirically assessed the relationship between the size of the application components and the delivered quality measured through user satisfaction. We have determined that, up to a certain extent, the average component size of an application is negatively related to the user satisfaction for
this application.

Discussed [29] web Performance testing is executed through testing campaigns for stressing the web site and back-end systems with the amount of load simulating the real conditions of the field or to evaluate if the site/application will support the expected load following special situations (e.g. an advertisement campaign). That allows you to guarantee system performances under that load and to identify and help in fixing possible issues.

Web performance measurement aims at analysis and fast characterization of system and user behaviour in order to give fast feedback on any issues. In order to achieve that, TILAB has realized two tools: WEBSAT(WEB Server Application Time) and BMPOP. WEBSAT is based on web server log files and completes the tool suite of the commercial off the shelf products used for Web Performance Evaluation/Measurement activities.

BMPOP (BenchMark Point Of Presence) is used for web performance measurement from the end-to-end perspective.

The study [30] reported in this paper establishes a conceptual framework and some key initial results in the analysis of the characteristics of software quality.

Explicit attention to characteristics of software quality can lead to significant savings in software life-cycle costs and the current software state-of-the-art imposes specific limitations on our ability to automatically and quantitatively evaluate the quality of software.

Proposed [31] a performance testing methodology which copes with the afore mentioned characteristics. the methodology consists of a set of methods and patterns to realize adequate workloads for multi-service systems. the effectiveness of this methodology is demonstrated throughout a case study on IP Multimedia Subsystem performance testing.

In [32] analyzed source code version control system logs of popular open source software systems to detect changes marked as refactorings and examine how the software metrics are affected by this process, in order to evaluate whether refactoring is effectively used as a means to improve software quality within the open source community.

Suggested [33] a framework for performance analysis and performance evaluation. So, we have discussed comparative analysis of various software performance testing tools and their limitations and a new approach for software performance testing. With an upbeat approach, we can produce software that can meet performance objectives and can be delivered on time and within budget, avoiding the project crises.

2.9 Summary

This chapter presents the theoretical information about what good code quality means, unit testing, dynamic testing, and continuous integration. Secondly, in next section describe the work related to software testing published by various researchers by highlighting their contributions.

Chapter Three

Methodology

Chapter Three
Methodology

3.1 Introduction

This chapter provides the research method steps, and the explanation of the methodology to be followed to achieve the research objectives. In addition, listing concrete tools and methods for improving code quality.

3.2 Research Methodologies and Software tool used

To develop ways to improve code quality using automated software testing tools in development projects, first, existing knowledge about code quality was investigated mostly through literature reviews and how to improve and implement it will be explored. Secondly, implement the knowledge about what things enhancement of code quality by using different tools and framework, finally, the methods and tools are put into framework of software development and the technologies are explained in more detail together with concrete examples of how to use them [19].

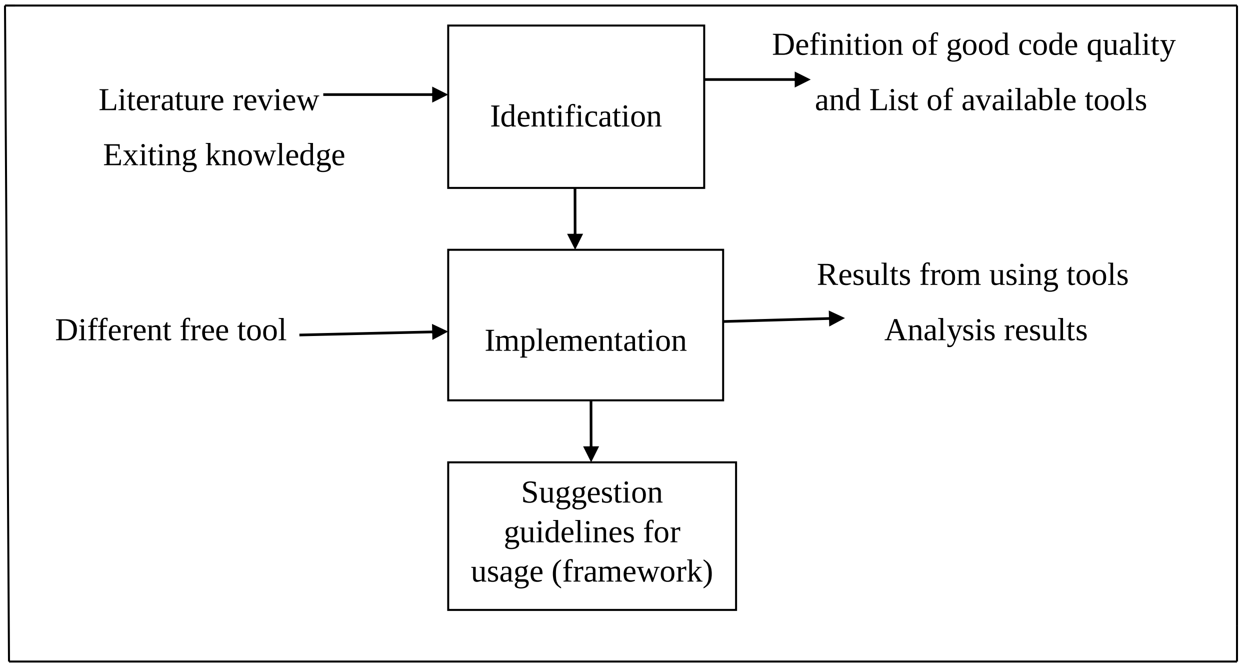


Figure 3.1: Research method for the study using Block diagram

3.2.1 Load test

WebLOAD Cloud (Web Dashboard in earlier versions). The Dashboards tab enables viewing, analyzing and comparing load sessions, with full control and customization of the display.

WebLOAD Cloud provides a single unified command and control interface where can create, execute, schedule, and analyze tests all directly from a web browser. It can be deployed on-premises or in the cloud, delivering greater visibility, control and collaboration to performance testing, QA, and DevOps teams. Some of the tasks can accomplish using the WebLOAD Cloud include:

- Create and edit load tests
- Upload, add and manage tests, resources, and sessions
- Execute and schedule test runs
- Analyze results using ready-made and self-configured dashboards
- Share tests and results with peers
- Manage access permissions for users and groups
- Download test results to the local machine for further analysis with WebLOAD Analytics

3.2.2 Apache JMeter

The Apache JMeter TM is pure Java open source software, which was first developed by Stefano Mazzocchi of the Apache Software Foundation, designed to load functional test behaviour and measure performance.

Used JMeter to analyze and measure the performance of web application or a variety of services. Performance Testing means testing a web application against heavy load, multiple and concurrent user traffic. JMeter originally is used for testing Web Application or FTP application. Nowadays, it is used for a functional test, database server test [35].

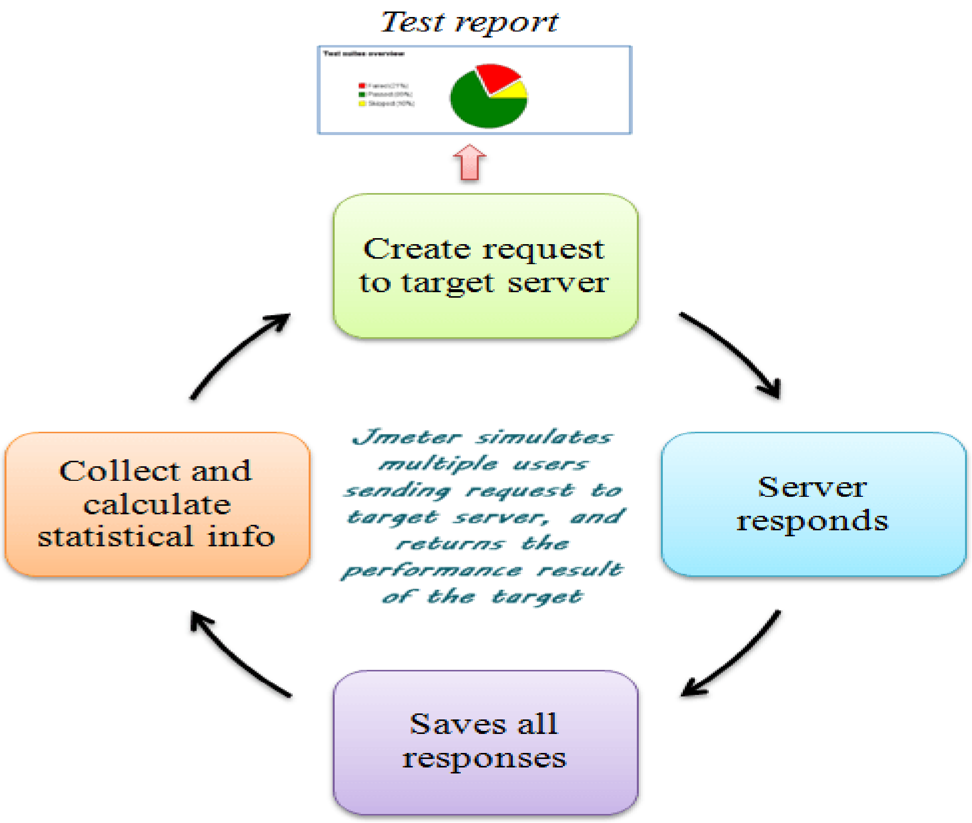
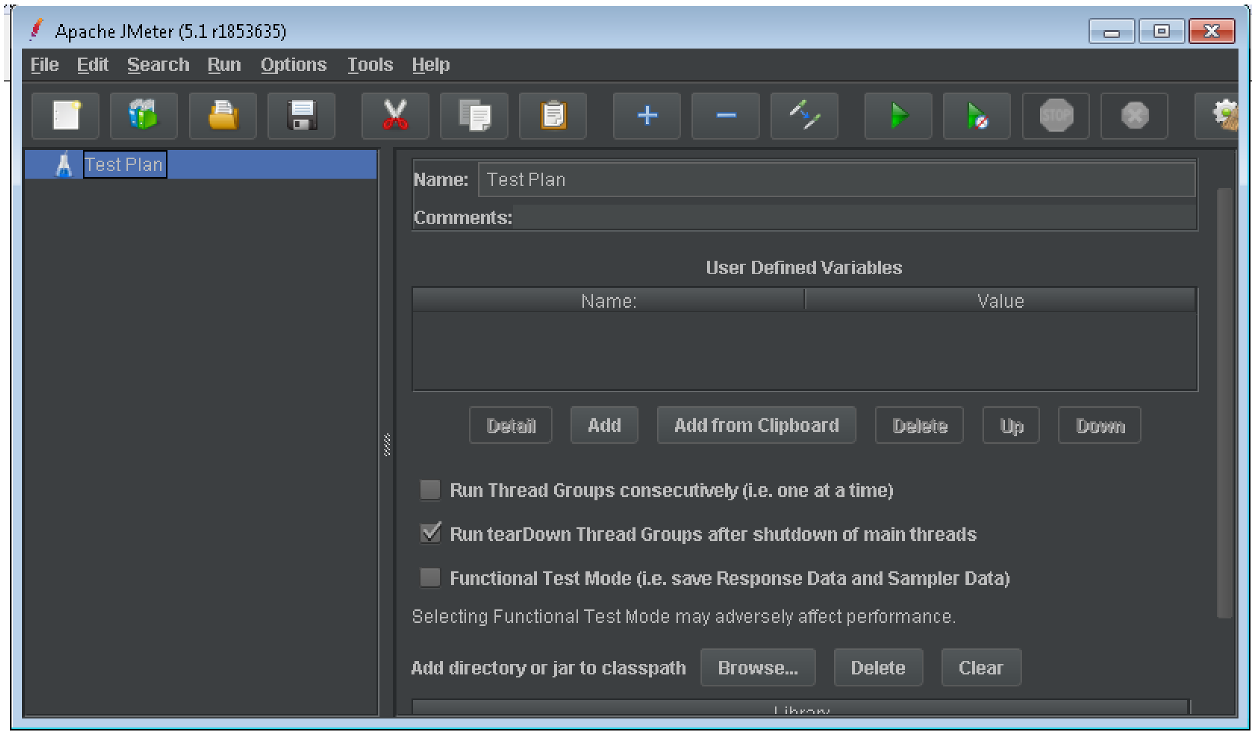

Test report
Create request to target server
Server responds
Saves all responses
Collect and calculate statistical info
Jmeter simulates multiple users sending request to target server, and returns the performance result of the target

Figure 3.2: Workflow of JMeter stages



Name: Test Plan

Comments:

User Defined Variables

Name:	Value

[ ] Run Thread Groups consecutively (i.e. one at a time)

[✓] Run tearDown Thread Groups after shutdown of main threads

[ ] Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

Add directory or jar to classpath

Figure 3.3: Home screen Apache JMeter

3.2.3 Unit testing using Jasmine

Jasmine is one of the popular JavaScript unit testing frameworks which is capable of testing synchronous and asynchronous JavaScript code. It is used in BDD (behaviour-driven development) programming which focuses more on the business value than on the technical details. Jasmine is a Behavior Driven Development testing framework for JavaScript. It does not rely on browsers, DOM, or any JavaScript framework. Thus it's suited for websites, Node.js projects, or anywhere that JavaScript can run. Jasmine Suite and Specs in Jasmine, there are two important terms – suite and spec.

3.2.3.1 Suite

A Jasmine suite is a group of test cases that can be used to test a specific behaviour of the JavaScript code (a JavaScript object or function). This begins with a call to the Jasmine global function describe with two parameters – the first parameter represents the title of the test suite and the second parameter represents a function that implements the test suite.

3.2.3.2 Spec

A Jasmine spec represents a test case inside the test suite. This begins with a call to the Jasmine global function it with two parameters – the first parameter represents the title of the spec and the second parameter represents a function that implements the test case.

In practice, spec contains one or more expectations. Each expectation represents an assertion that can be either true or false. To pass the spec, all of the expectations inside the spec have to be true. If one or more expectations inside a spec are false, the spec fails [36].

3.3 Summary

This chapter provides a description of the research methodology using a block diagram. Also, show up the techniques have been applied in the research, in addition describe different tools and steps of a method used.

Chapter Four

Implementation and Results

Chapter Four

Implementation and Results

4.1 Introduction

This chapter shows the implementation details of code quality in web pages using software testing tool. It also shows the results of the study are presented and discussed concerning the aim of the study, which was to provide framework or suggestion for how to design, implement and administer tools for code quality in software development. First we applied tools that improve performance website, secondly, when the code is in operation mode, unit testing using jasmine framework is performed in the runtime environment.

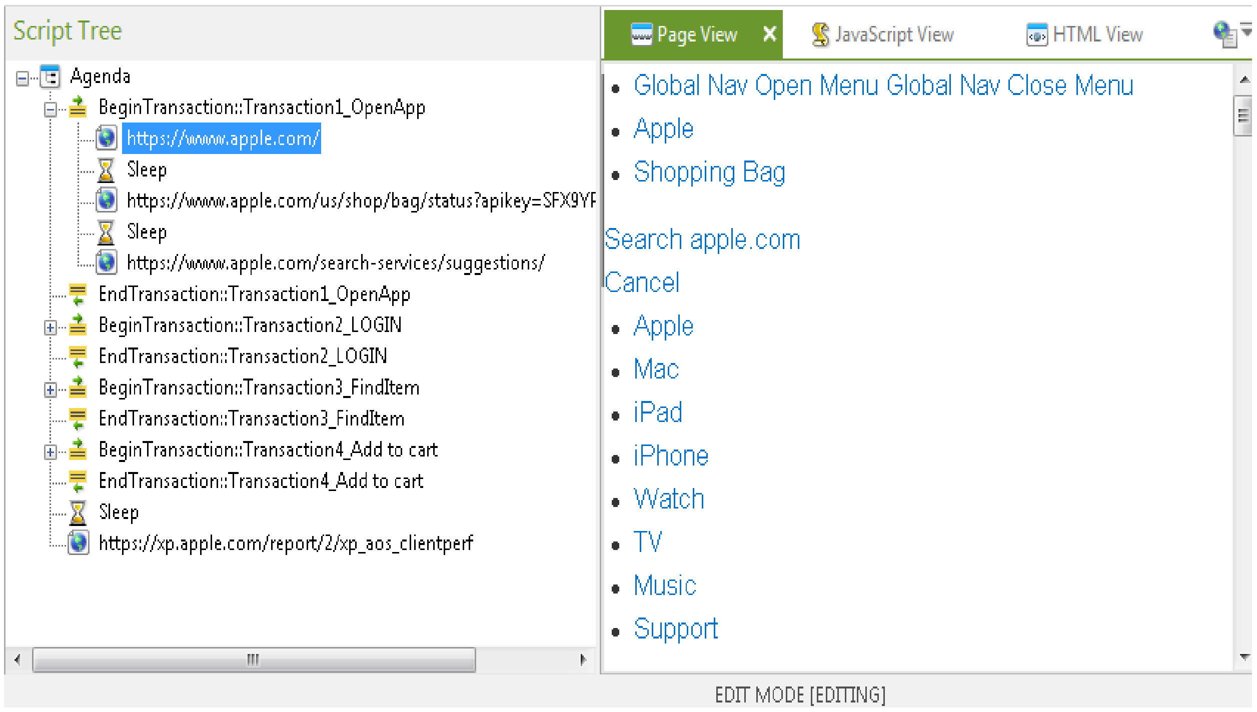
4.2 Performance testing with WebLOAD Professional

In this study, three sites were tested in order to get more coverage of the results and compare them with another tool using the same sites. The sites used in this study are:

Apple.com and pstore.com this site is designed by me and the interactive social site noonpost.com.

4.2.1 First site https://www.apple.com/

The first steps to test the performance of the webload tool, WebLOAD Recorder. Is a visual environment for creating protocol test scripts (referred to as scripts). The snapshot of recorded agenda can switch the views from JavaScript View to Page View or HTML View or HTTP Headers View.



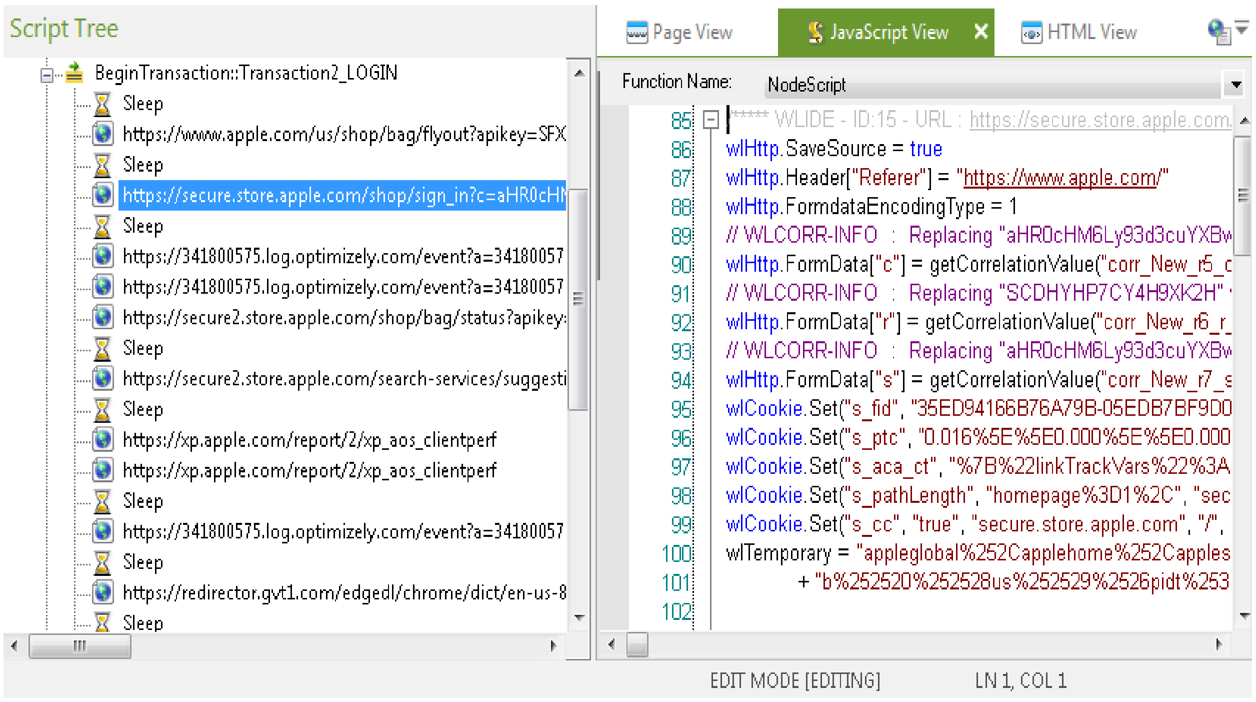
```
Agenda
  BeginTransaction::Transaction1_OpenApp
    https://www.apple.com/
    Sleep
    https://www.apple.com/us/shop/bag/status?apikey=SFX9YF
    Sleep
    https://www.apple.com/search-services/suggestions/
  EndTransaction::Transaction1_OpenApp
  BeginTransaction::Transaction2_LOGIN
  EndTransaction::Transaction2_LOGIN
  BeginTransaction::Transaction3_FindItem
  EndTransaction::Transaction3_FindItem
  BeginTransaction::Transaction4_Add to cart
  EndTransaction::Transaction4_Add to cart
  Sleep
  https://xp.apple.com/report/2/xp_aos_clientperf
```

- Global Nav Open Menu Global Nav Close Menu
- Apple
- Shopping Bag

Search apple.com
Cancel

- Apple
- Mac
- iPad
- iPhone
- Watch
- TV
- Music
- Support

Figure 4.1: Agenda for (apple.com) and Page View

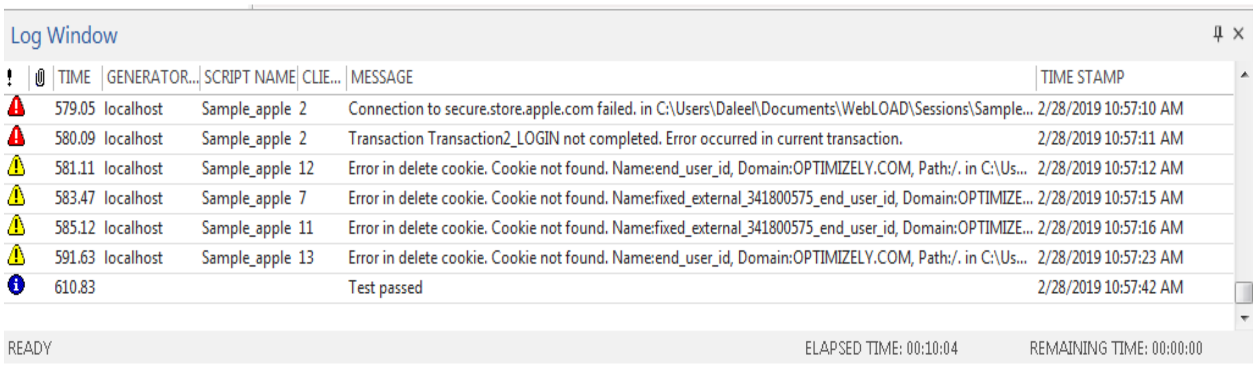


```
BeginTransaction::Transaction2_LOGIN
  Sleep
  https://www.apple.com/us/shop/bag/flyout?apikey=SFX
  Sleep
  https://secure.store.apple.com/shop/sign_in?c=aHR0cHI
  Sleep
  https://341800575.log.optimizely.com/event?a=34180057
  https://341800575.log.optimizely.com/event?a=34180057
  https://secure2.store.apple.com/shop/bag/status?apikey:
  Sleep
  https://secure2.store.apple.com/search-services/suggesti
  Sleep
  https://xp.apple.com/report/2/xp_aos_clientperf
  https://xp.apple.com/report/2/xp_aos_clientperf
  Sleep
  https://341800575.log.optimizely.com/event?a=34180057
  Sleep
  https://redirector.gvt1.com/edgedl/chrome/dict/en-us-8
  Sleep
```

Function Name: NodeScript

```
/***** WLIDE - ID:15 - URL : https://secure.store.apple.com.
wlHttp.SaveSource = true
wlHttp.Header["Referer"] = "https://www.apple.com/"
wlHttp.FormdataEncodingType = 1
// WLCORR-INFO  :  Replacing "aHR0cHM6Ly93d3cuYXBw
wlHttp.FormData["c"] = getCorrelationValue("corr_New_r5_c
// WLCORR-INFO  :  Replacing "SCDHYHP7CY4H9XK2H"
wlHttp.FormData["r"] = getCorrelationValue("corr_New_r6_r_
// WLCORR-INFO  :  Replacing "aHR0cHM6Ly93d3cuYXBw
wlHttp.FormData["s"] = getCorrelationValue("corr_New_r7_s
wlCookie.Set("s_fid", "35ED94166B76A79B-05EDB7BF9D0
wlCookie.Set("s_ptc", "0.016%5E%5E0.000%5E%5E0.000
wlCookie.Set("s_aca_ct", "%7B%22linkTrackVars%22%3A
wlCookie.Set("s_pathLength", "homepage%3D1%2C", "sec
wlCookie.Set("s_cc", "true", "secure.store.apple.com", "/",
wlTemporary = "appleglobal%252Capplehome%252Capples
        + "b%252520%252528us%252529%2526pidt%253
```

Figure 4.2: Agenda for(apple.com) and Javascript View

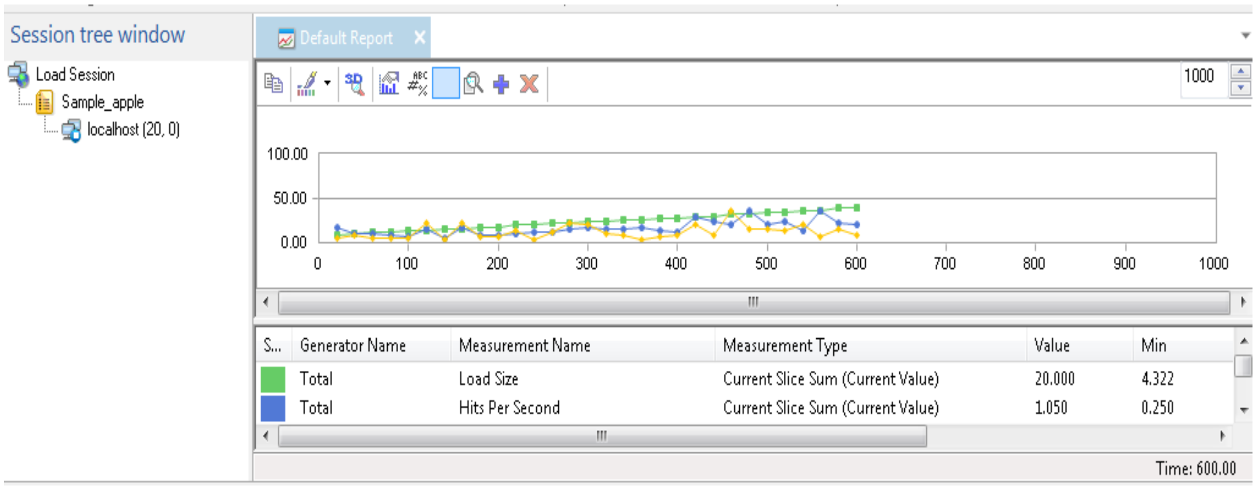

Log Window

!	📎	TIME	GENERATOR...	SCRIPT NAME	CLIE...	MESSAGE	TIME STAMP
⚠		579.05	localhost	Sample_apple	2	Connection to secure.store.apple.com failed. in C:\Users\Daleel\Documents\WebLOAD\Sessions\Sample...	2/28/2019 10:57:10 AM
⚠		580.09	localhost	Sample_apple	2	Transaction Transaction2_LOGIN not completed. Error occurred in current transaction.	2/28/2019 10:57:11 AM
⚠		581.11	localhost	Sample_apple	12	Error in delete cookie. Cookie not found. Name:end_user_id, Domain:OPTIMIZELY.COM, Path:/. in C:\Us...	2/28/2019 10:57:12 AM
⚠		583.47	localhost	Sample_apple	7	Error in delete cookie. Cookie not found. Name:fixed_external_341800575_end_user_id, Domain:OPTIMIZE...	2/28/2019 10:57:15 AM
⚠		585.12	localhost	Sample_apple	11	Error in delete cookie. Cookie not found. Name:fixed_external_341800575_end_user_id, Domain:OPTIMIZE...	2/28/2019 10:57:16 AM
⚠		591.63	localhost	Sample_apple	13	Error in delete cookie. Cookie not found. Name:end_user_id, Domain:OPTIMIZELY.COM, Path:/. in C:\Us...	2/28/2019 10:57:23 AM
ℹ		610.83				Test passed	2/28/2019 10:57:42 AM

READY ELAPSED TIME: 00:10:04 REMAINING TIME: 00:00:00

Figure 4.3: Log Window (apple.com)

The Log Window displays a summary of the test including all log messages detected by WebLOAD Console in run time that is generated by the Console, the JavaScript compiler and any user messages programmed in the test script.

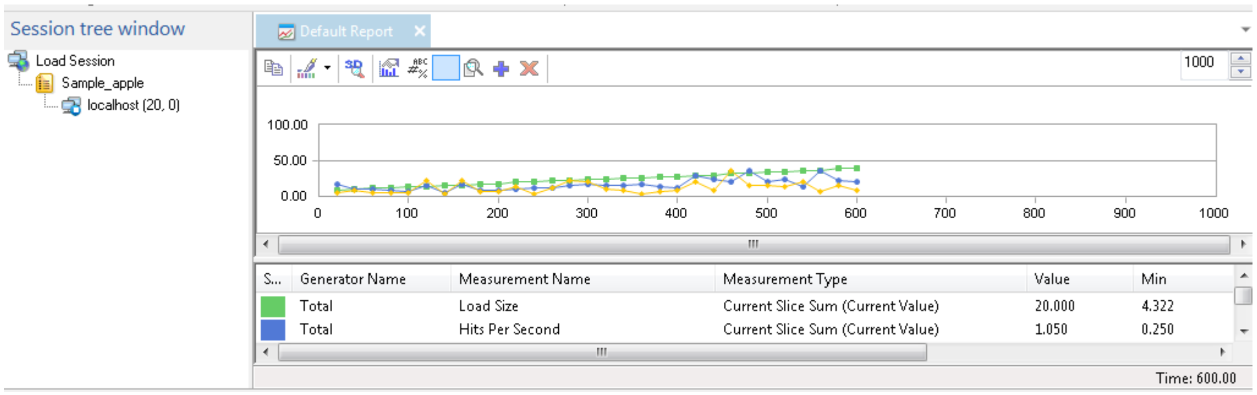


S...	Generator Name	Measurement Name	Measurement Type	Value	Min
🟩	Total	Load Size	Current Slice Sum (Current Value)	20.000	4.322
🟦	Total	Hits Per Second	Current Slice Sum (Current Value)	1.050	0.250

Time: 600.00

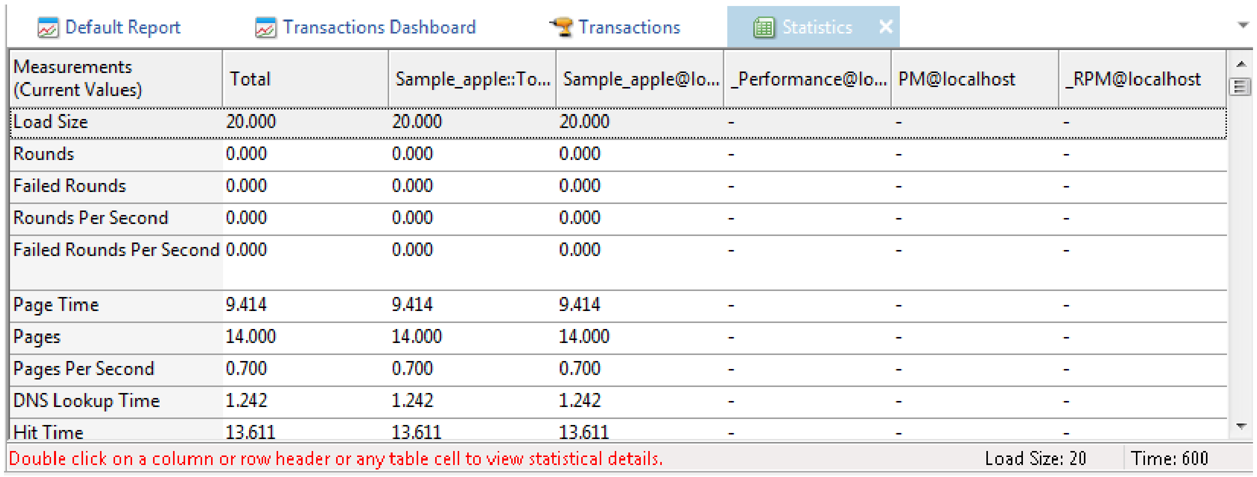
Figure 4.4: Session Tree and default report with measurement type (apple.com)

The Session Tree displays in the left pane of the Console screen, and gives you a complete graphical overview of the test session including the Agendas run, and the hosts running each Agenda. The icons adjacent to the tree items enable you to view your test activity at a glance.



S...	Generator Name	Measurement Name	Measurement Type	Value	Min
🟩	Total	Load Size	Current Slice Sum (Current Value)	20.000	4.322
🟦	Total	Hits Per Second	Current Slice Sum (Current Value)	1.050	0.250

Figure 4.5: Default Report (apple.com)

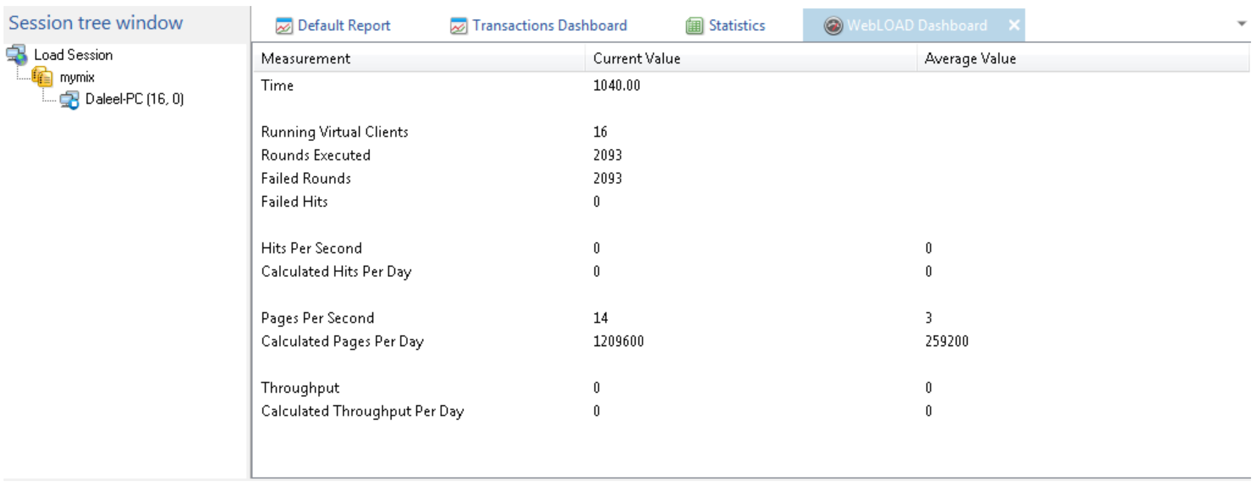


Measurements (Current Values)	Total	Sample_apple::To...	Sample_apple@lo...	_Performance@lo...	PM@localhost	_RPM@localhost
Load Size	20.000	20.000	20.000	-	-	-
Rounds	0.000	0.000	0.000	-	-	-
Failed Rounds	0.000	0.000	0.000	-	-	-
Rounds Per Second	0.000	0.000	0.000	-	-	-
Failed Rounds Per Second	0.000	0.000	0.000	-	-	-
Page Time	9.414	9.414	9.414	-	-	-
Pages	14.000	14.000	14.000	-	-	-
Pages Per Second	0.700	0.700	0.700	-	-	-
DNS Lookup Time	1.242	1.242	1.242	-	-	-
Hit Time	13.611	13.611	13.611	-	-	-

Double click on a column or row header or any table cell to view statistical details. Load Size: 20 Time: 600

Figure 4.6: Statistics Reports (apple.com)

WebLOAD collects approximately 35 different statistics during a test. Statistics Reports display the values for all of them.



Measurement	Current Value	Average Value
Time	1040.00	
Running Virtual Clients	16	
Rounds Executed	2093	
Failed Rounds	2093	
Failed Hits	0	
Hits Per Second	0	0
Calculated Hits Per Day	0	0
Pages Per Second	14	3
Calculated Pages Per Day	1209600	259200
Throughput	0	0
Calculated Throughput Per Day	0	0

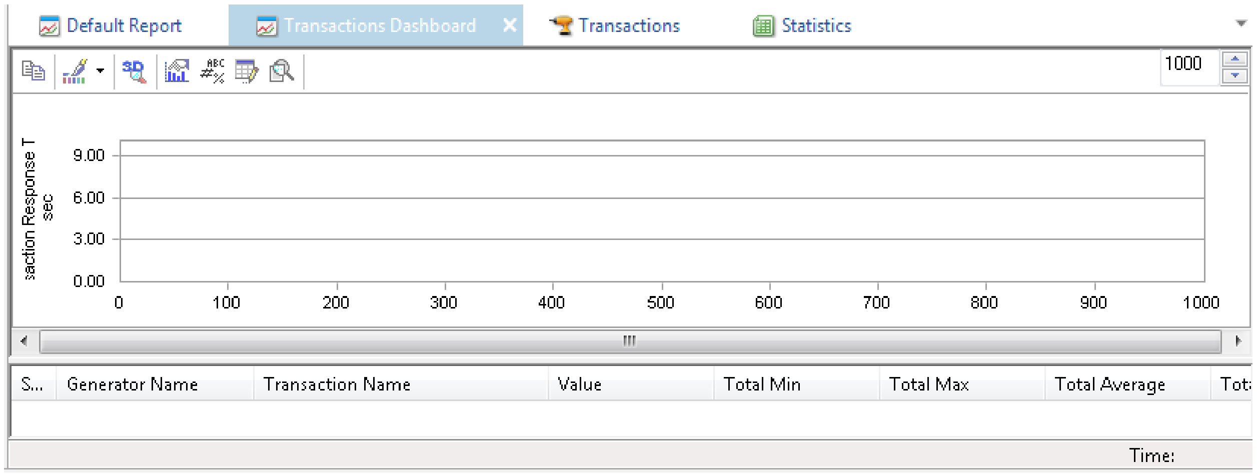


Figure 4.7: Dashboard (apple.com)

The Dashboard displays real-time statistical information about the test session including the number of Virtual Clients running, hits per day, pages per day and throughput.

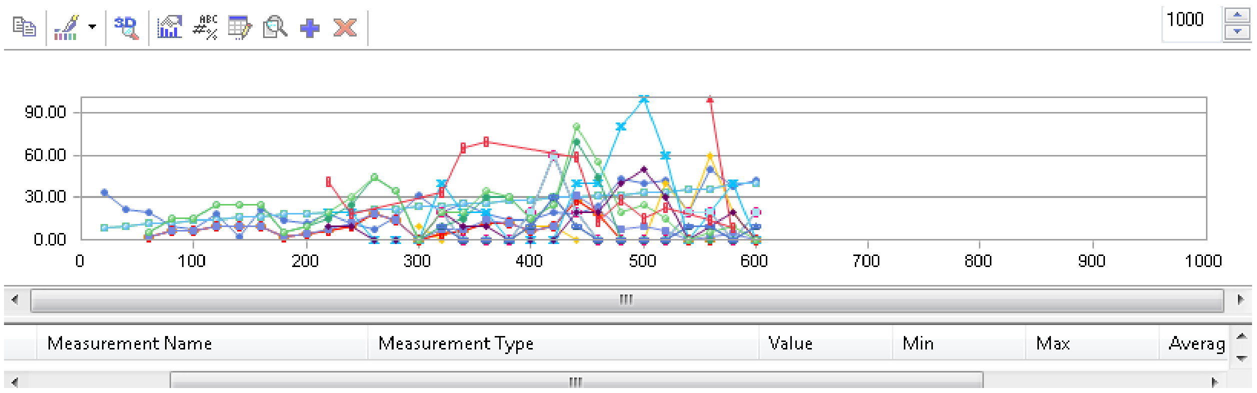
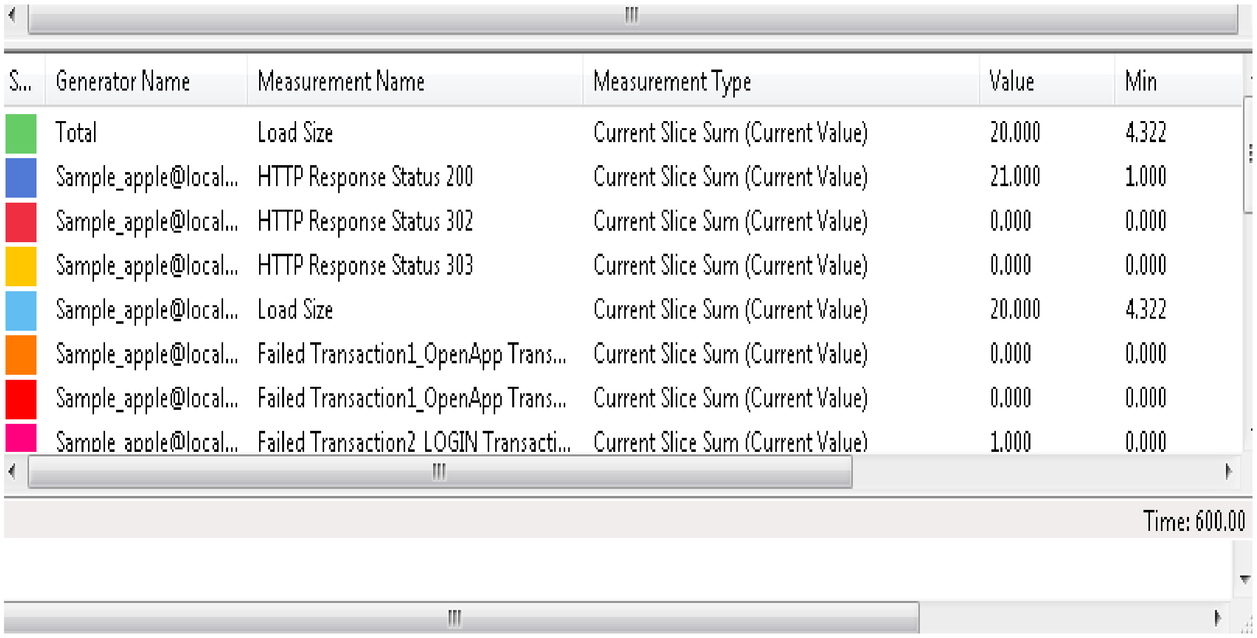


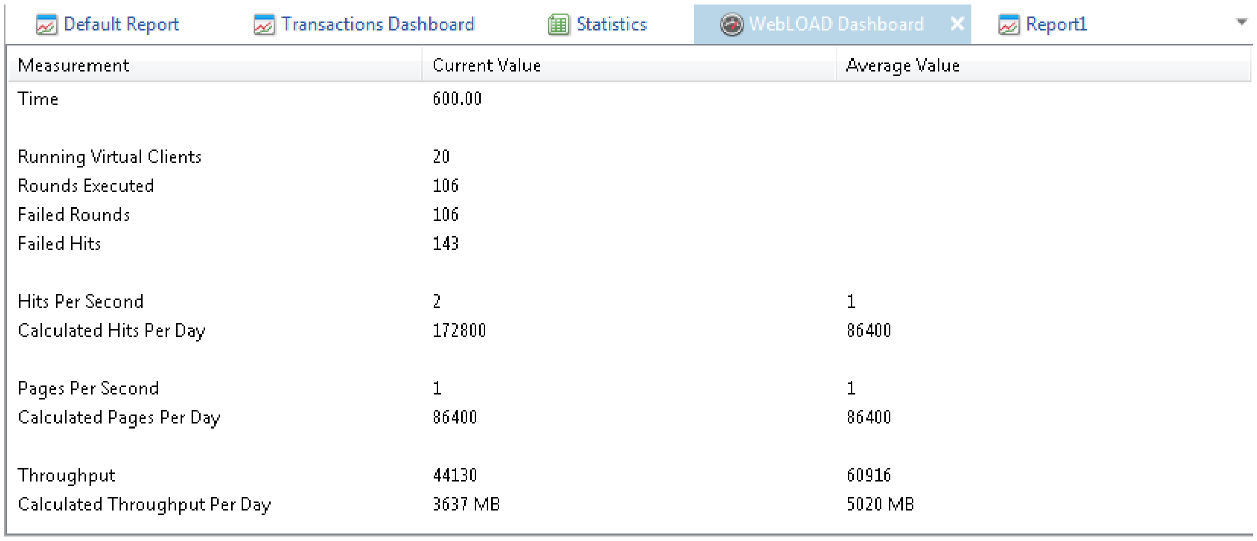
Figure 4.8: Integrated Report (apple.com)



S...	Generator Name	Measurement Name	Measurement Type	Value	Min
🟩	Total	Load Size	Current Slice Sum (Current Value)	20.000	4.322
🟦	Sample_apple@local...	HTTP Response Status 200	Current Slice Sum (Current Value)	21.000	1.000
🟥	Sample_apple@local...	HTTP Response Status 302	Current Slice Sum (Current Value)	0.000	0.000
🟨	Sample_apple@local...	HTTP Response Status 303	Current Slice Sum (Current Value)	0.000	0.000
🟦	Sample_apple@local...	Load Size	Current Slice Sum (Current Value)	20.000	4.322
🟧	Sample_apple@local...	Failed Transaction1_OpenApp Trans...	Current Slice Sum (Current Value)	0.000	0.000
🟥	Sample_apple@local...	Failed Transaction1_OpenApp Trans...	Current Slice Sum (Current Value)	0.000	0.000
🟪	Sample_apple@local...	Failed Transaction2_LOGIN Transacti...	Current Slice Sum (Current Value)	1.000	0.000

Time: 600.00

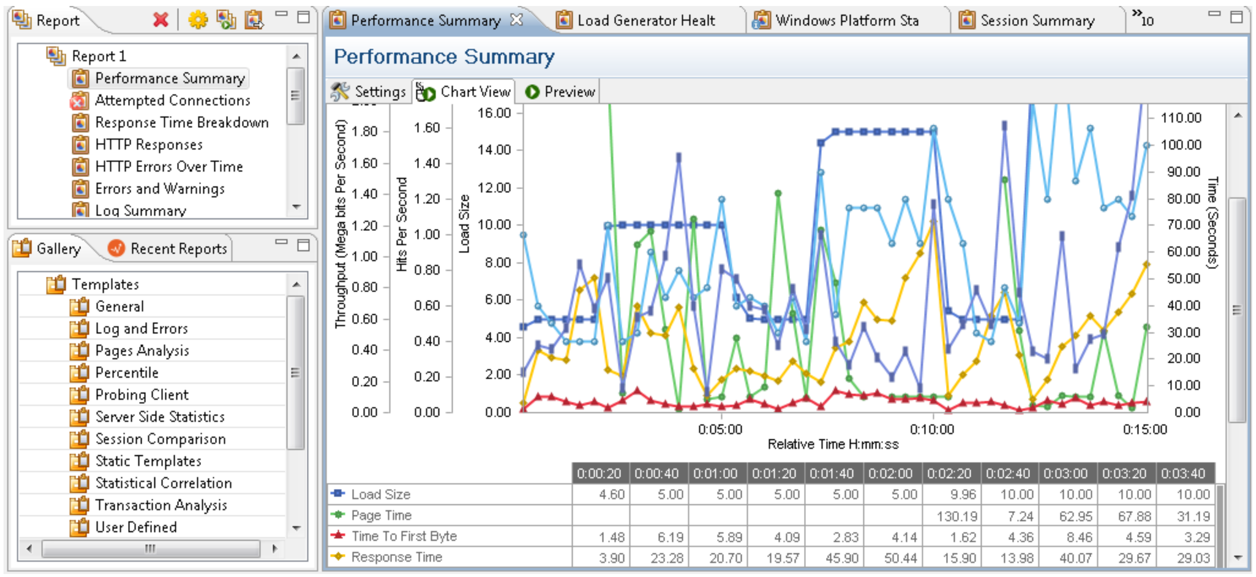
Figure 4.9: measurement type (apple.com/)



Measurement	Current Value	Average Value
Time	600.00	
Running Virtual Clients	20	
Rounds Executed	106	
Failed Rounds	106	
Failed Hits	143	
Hits Per Second	2	1
Calculated Hits Per Day	172800	86400
Pages Per Second	1	1
Calculated Pages Per Day	86400	86400
Throughput	44130	60916
Calculated Throughput Per Day	3637 MB	5020 MB

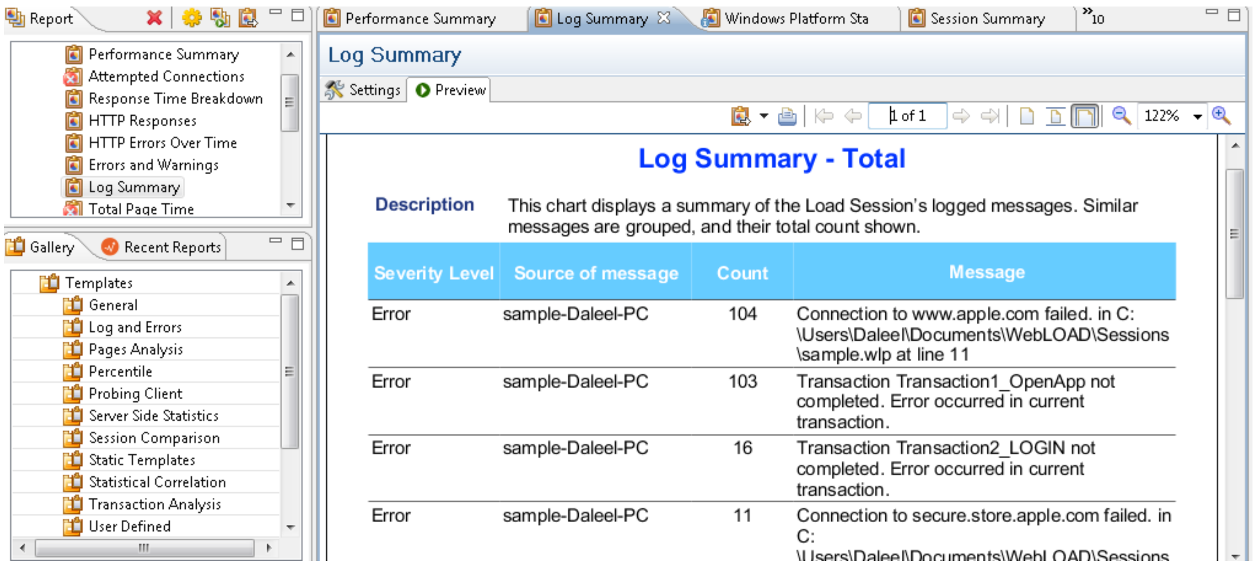
Figure 4.10: Dashboard (apple.com)

The last step, WebLOAD Analytics provides you with a simple yet comprehensive method of producing and publishing reports to fulfill all your analysis and reporting requirements. Using WebLOAD Analytics, you can create clear, accurate, and meaningful reports that enable you to analyze Load Session results and identify peaks, trends, and anomalies in your data. WebLOAD Analytics provides you with a variety of predefined chart templates that enable you to produce focused reports on specific topics. You can edit these templates, and create new templates.



	0:00:20	0:00:40	0:01:00	0:01:20	0:01:40	0:02:00	0:02:20	0:02:40	0:03:00	0:03:20	0:03:40
Load Size	4.60	5.00	5.00	5.00	5.00	5.00	9.96	10.00	10.00	10.00	10.00
Page Time							130.19	7.24	62.95	67.88	31.19
Time To First Byte	1.48	6.19	5.89	4.09	2.83	4.14	1.62	4.36	8.46	4.59	3.29
Response Time	3.90	23.28	20.70	19.57	45.90	50.44	15.90	13.98	40.07	29.67	29.03

Figure 4.11: WebLOAD Analytics Screen (apple.com)

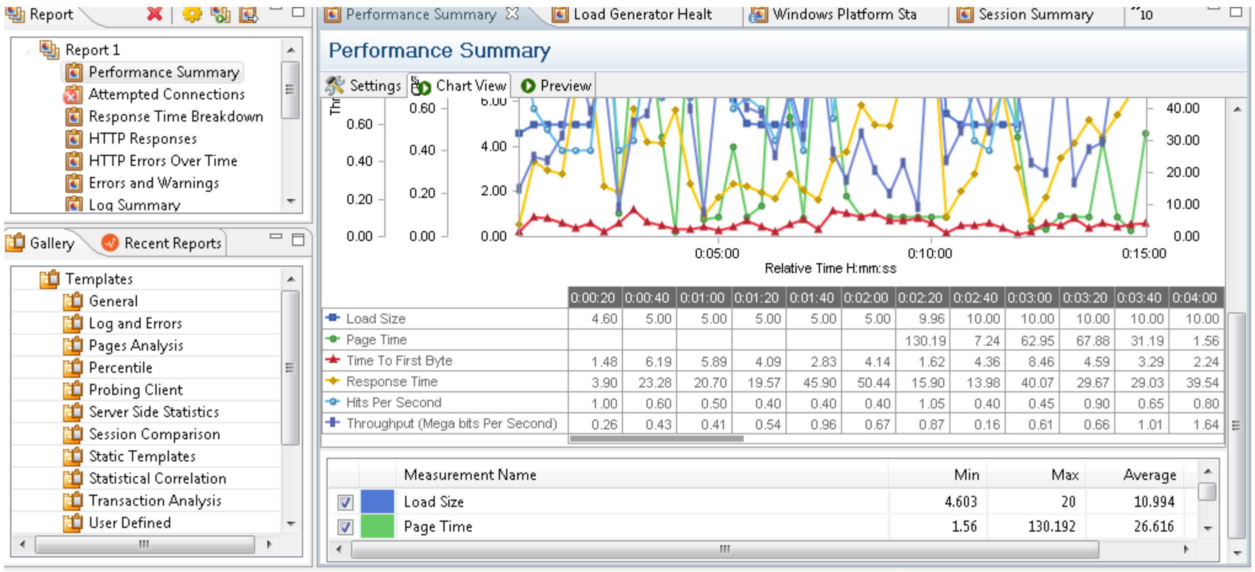


Log Summary - Total

Description This chart displays a summary of the Load Session's logged messages. Similar messages are grouped, and their total count shown.

Severity Level	Source of message	Count	Message
Error	sample-Daleel-PC	104	Connection to www.apple.com failed. in C: \Users\Daleel\Documents\WebLOAD\Sessions \sample.wlp at line 11
Error	sample-Daleel-PC	103	Transaction Transaction1_OpenApp not completed. Error occurred in current transaction.
Error	sample-Daleel-PC	16	Transaction Transaction2_LOGIN not completed. Error occurred in current transaction.
Error	sample-Daleel-PC	11	Connection to secure.store.apple.com failed. in C: \Users\Daleel\Documents\WebLOAD\Sessions

Figure 4.12: Log Summary, Tabular Data (apple.com)



	0:00:20	0:00:40	0:01:00	0:01:20	0:01:40	0:02:00	0:02:20	0:02:40	0:03:00	0:03:20	0:03:40	0:04:00
Load Size	4.60	5.00	5.00	5.00	5.00	5.00	9.96	10.00	10.00	10.00	10.00	10.00
Page Time							130.19	7.24	62.95	67.88	31.19	1.56
Time To First Byte	1.48	6.19	5.89	4.09	2.83	4.14	1.62	4.36	8.46	4.59	3.29	2.24
Response Time	3.90	23.28	20.70	19.57	45.90	50.44	15.90	13.98	40.07	29.67	29.03	39.54
Hits Per Second	1.00	0.60	0.50	0.40	0.40	0.40	1.05	0.40	0.45	0.90	0.65	0.80
Throughput (Mega bits Per Second)	0.26	0.43	0.41	0.54	0.96	0.67	0.87	0.16	0.61	0.66	1.01	1.64

		Measurement Name	Min	Max	Average
✓	🟦	Load Size	4.603	20	10.994
✓	🟩	Page Time	1.56	130.192	26.616

Figure 4.13: WebLOAD Analytics User Interface (apple.com)

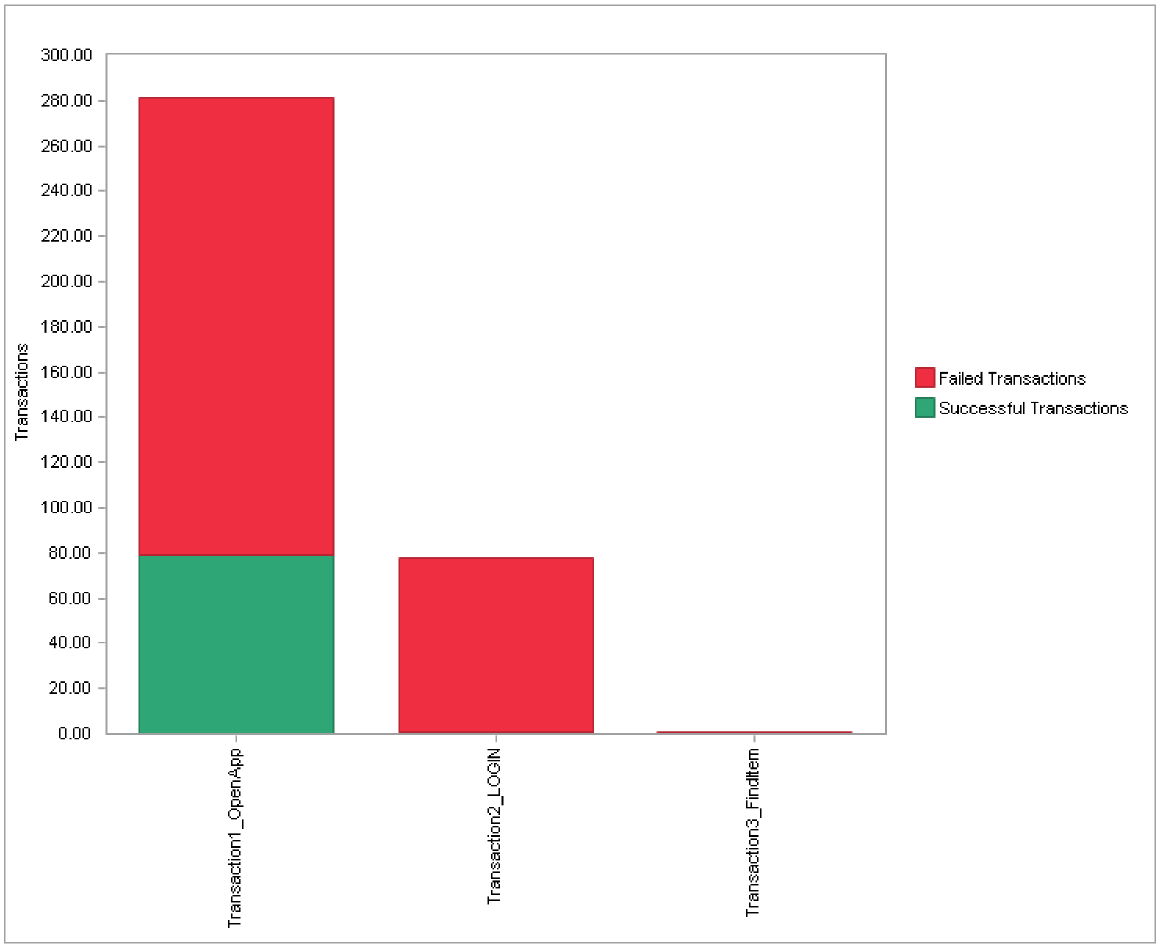


Figure 4.14: Transaction Counters (apple.com)

Measurement Name	Type	Total
Successful Transaction1_OpenApp Transactions	Sum	79
Successful Transaction2_LOGIN Transactions	Sum	1
Failed Transaction1_OpenApp Transactions	Sum	202
Failed Transaction2_LOGIN Transactions	Sum	77
Failed Transaction3_FindItem Transactions	Sum	1

Table 1: Measurement based Transaction Counters(apple.com)

This chart shows successful and failed transaction counts for each transaction in the session.

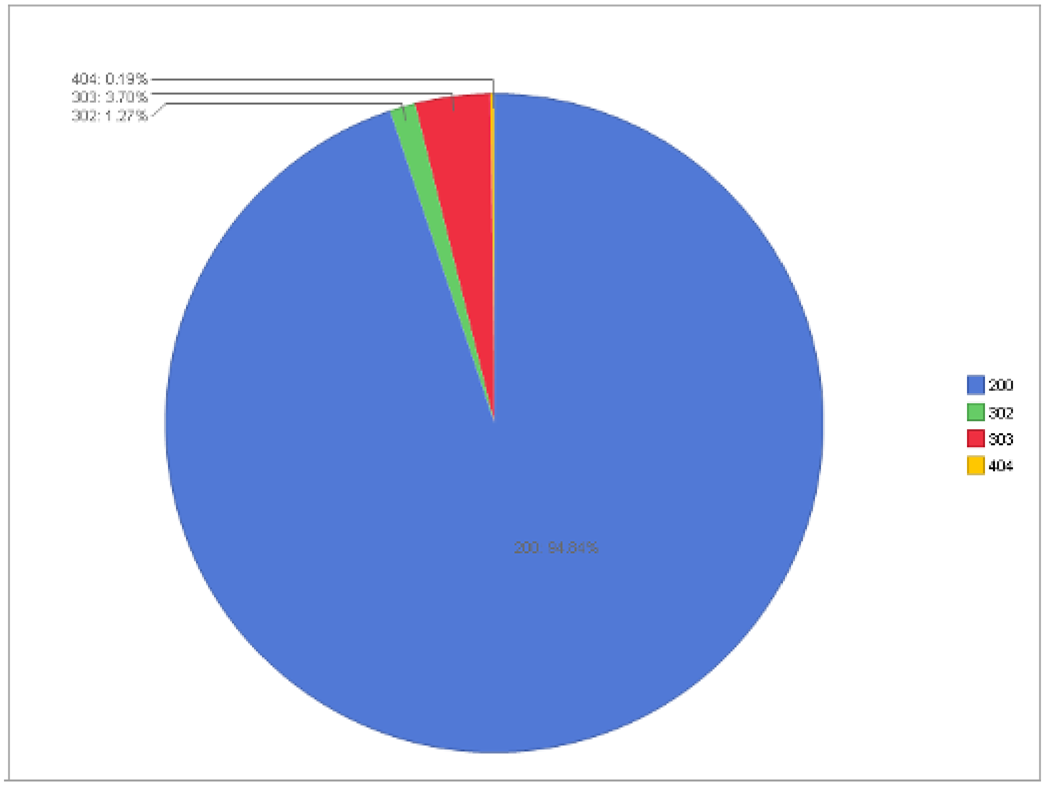


Figure 4.15: HTTP Responses(apple.com)

Measurement Name	Type	Total
HTTP Response Status 200	Sum	974
HTTP Response Status 302	Sum	13
HTTP Response Status 303	Sum	38
HTTP Response Status 404	Sum	2

Table 2: Measurement based HTTP Responses(apple.com)

This chart displays a summary of the HTTP response status messages received during the Load Session. For each response status, the template lists the number of responses received and what percentage it represents of all HTTP responses. Common response status codes - 200 OK, 302 Found, 404 Not Found, 500 Internal Server Error.

4.2.2 Second site http://www.noonpost.com/

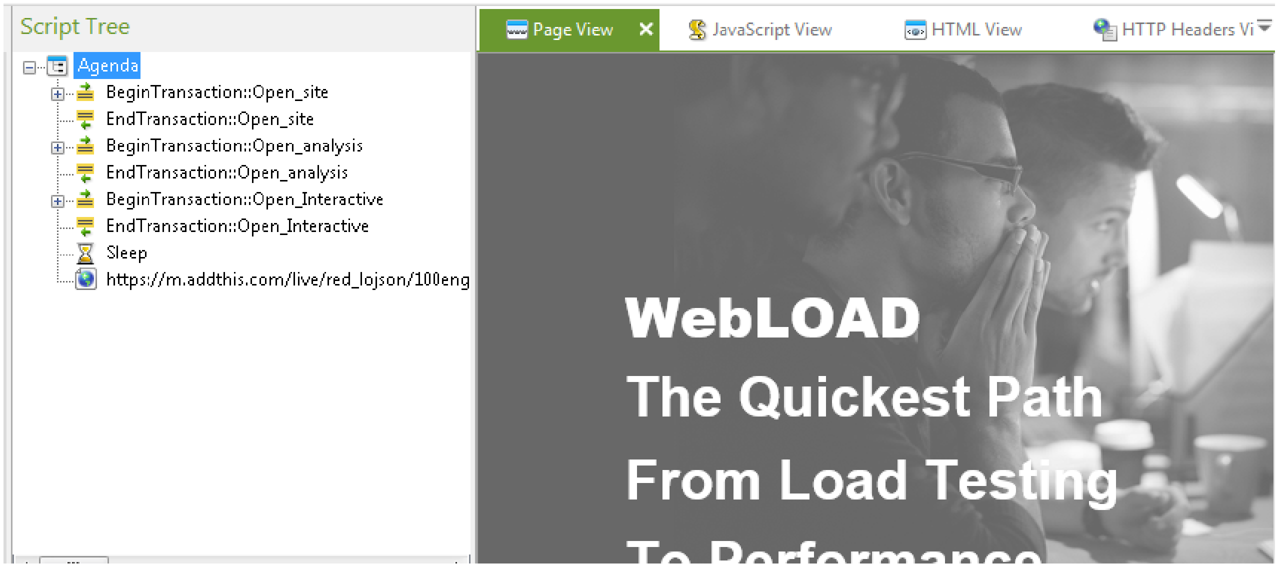
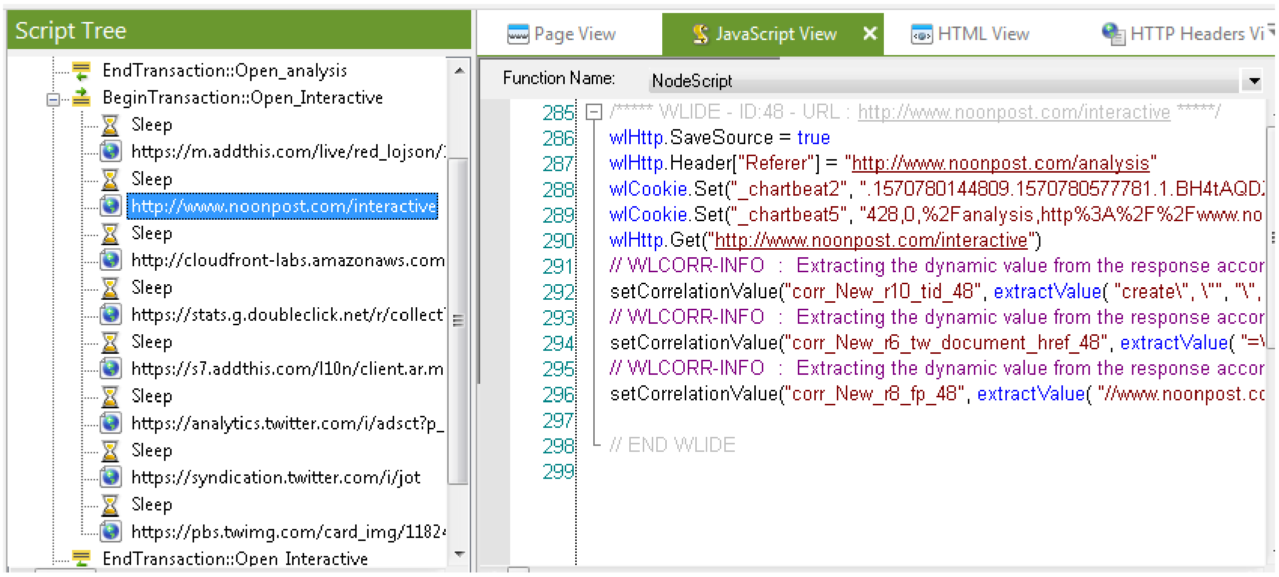

Script Tree
Agenda
BeginTransaction::Open_site
EndTransaction::Open_site
BeginTransaction::Open_analysis
EndTransaction::Open_analysis
BeginTransaction::Open_Interactive
EndTransaction::Open_Interactive
Sleep
https://m.addthis.com/live/red_lojson/100eng
Page View
JavaScript View
HTML View
HTTP Headers Vi
WebLOAD
The Quickest Path
From Load Testing
To Performance

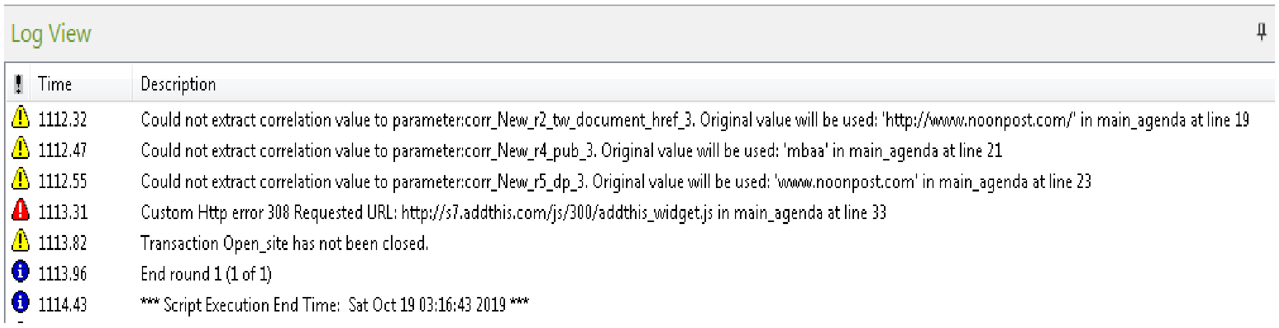
Figure 4.16: Agenda for(noonpost) and Page View



```
EndTransaction::Open_analysis
BeginTransaction::Open_Interactive
    Sleep
    https://m.addthis.com/live/red_lojson/1
    Sleep
    http://www.noonpost.com/interactive
    Sleep
    http://cloudfront-labs.amazonaws.com
    Sleep
    https://stats.g.doubleclick.net/r/collect
    Sleep
    https://s7.addthis.com/l10n/client.ar.m
    Sleep
    https://analytics.twitter.com/i/adsct?p_
    Sleep
    https://syndication.twitter.com/i/jot
    Sleep
    https://pbs.twimg.com/card_img/1182
EndTransaction::Open_Interactive
```

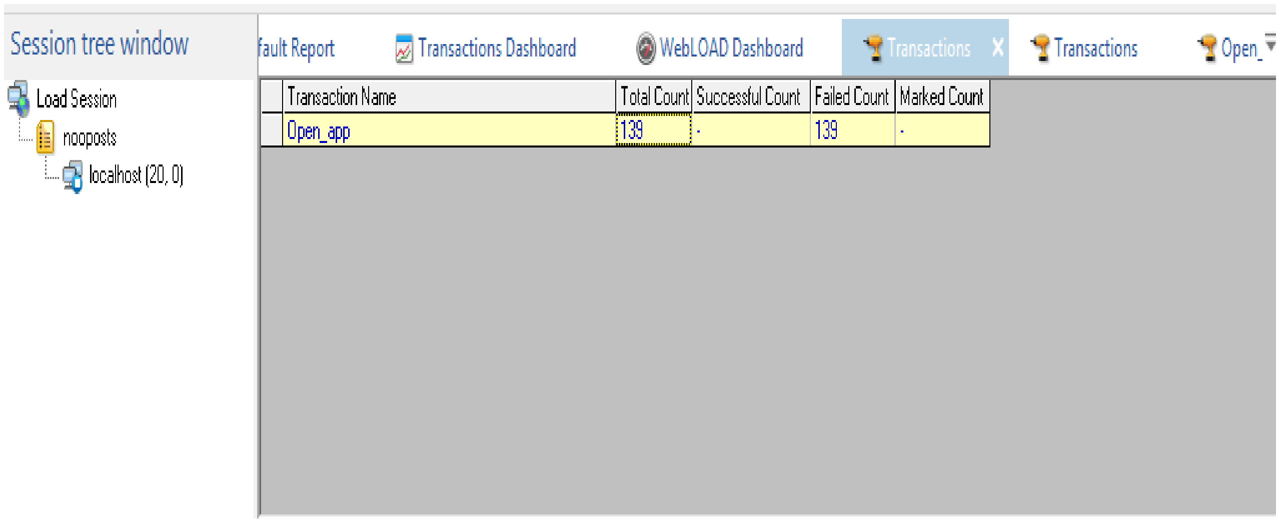
```
/***** WLIDE - ID:48 - URL : http://www.noonpost.com/interactive *****/
wlHttp.SaveSource = true
wlHttp.Header["Referer"] = "http://www.noonpost.com/analysis"
wlCookie.Set("_chartbeat2", ".1570780144809.1570780577781.1.BH4tAQD
wlCookie.Set("_chartbeat5", "428,0,%2Fanalysis,http%3A%2F%2Fwww.no
wlHttp.Get("http://www.noonpost.com/interactive")
// WLCORR-INFO  :  Extracting the dynamic value from the response accor
setCorrelationValue("corr_New_r10_tid_48", extractValue( "create\", \"", "\",
// WLCORR-INFO  :  Extracting the dynamic value from the response accor
setCorrelationValue("corr_New_r6_tw_document_href_48", extractValue( "=\
// WLCORR-INFO  :  Extracting the dynamic value from the response accor
setCorrelationValue("corr_New_r8_fp_48", extractValue( "//www.noonpost.cc

// END WLIDE
```

Figure 4.17: Agenda for(noonpost) and Javascript View



```
!  Time     Description
⚠  1112.32  Could not extract correlation value to parameter:corr_New_r2_tw_document_href_3. Original value will be used: 'http://www.noonpost.com/' in main_agenda at line 19
⚠  1112.47  Could not extract correlation value to parameter:corr_New_r4_pub_3. Original value will be used: 'mbaa' in main_agenda at line 21
⚠  1112.55  Could not extract correlation value to parameter:corr_New_r5_dp_3. Original value will be used: 'www.noonpost.com' in main_agenda at line 23
⚠  1113.31  Custom Http error 308 Requested URL: http://s7.addthis.com/js/300/addthis_widget.js in main_agenda at line 33
⚠  1113.82  Transaction Open_site has not been closed.
ℹ  1113.96  End round 1 (1 of 1)
ℹ  1114.43  *** Script Execution End Time:  Sat Oct 19 03:16:43 2019 ***
```

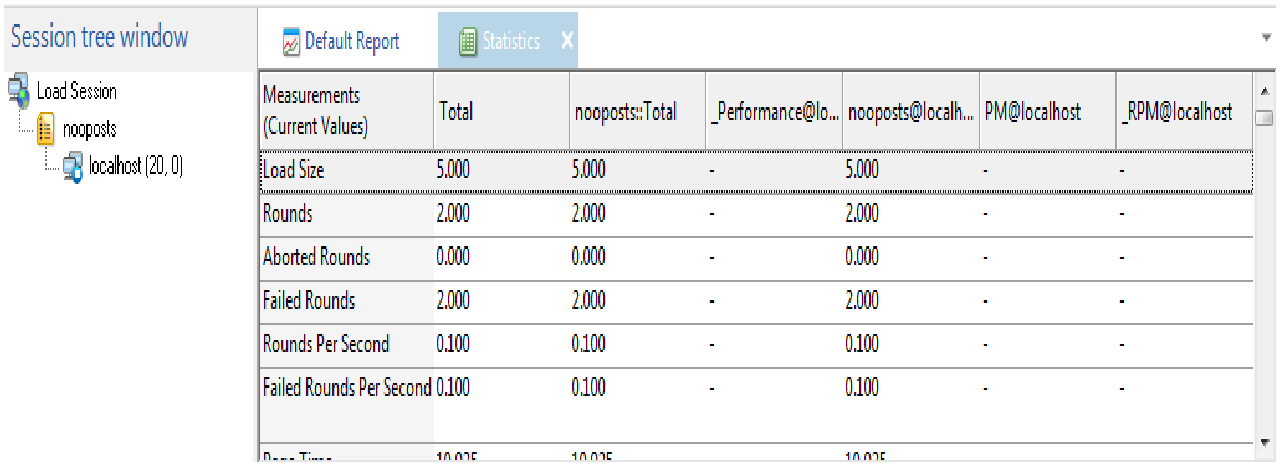
Figure 4.18: Log Window (noonpost)



Transaction Name	Total Count	Successful Count	Failed Count	Marked Count
Open_app	139	-	139	-

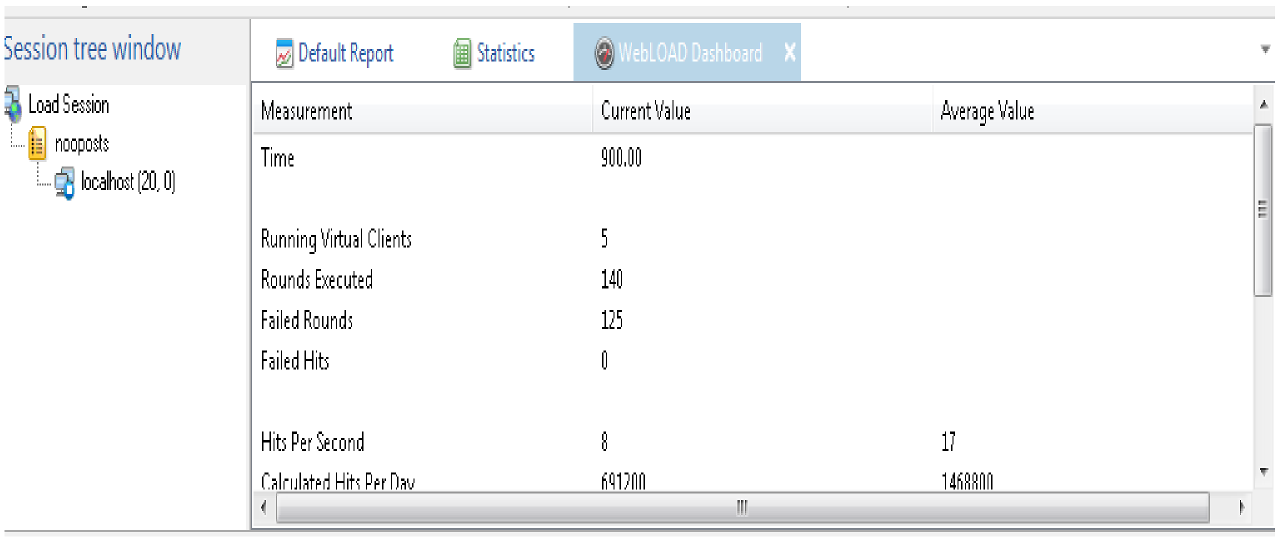
Figure 4.19: Data Drilling Report(noonpost)

Data Drilling provides both a global and detailed account of hit successes and failures allowing you to verify the functional integrity of your Web application at the per client, per-transactions and per-instance level. The Data Drilling reports provide an extremely detailed yet easily accessible summary of all the statistical, timing, and performance information collected over the course of the test session.



Measurements (Current Values)	Total	nooposts::Total	_Performance@lo...	nooposts@localh...	PM@localhost	_RPM@localhost
Load Size	5.000	5.000	-	5.000	-	-
Rounds	2.000	2.000	-	2.000	-	-
Aborted Rounds	0.000	0.000	-	0.000	-	-
Failed Rounds	2.000	2.000	-	2.000	-	-
Rounds Per Second	0.100	0.100	-	0.100	-	-
Failed Rounds Per Second	0.100	0.100	-	0.100	-	-

Figure 4.20: Statistics Reports(noonpost)


Session tree window

Load Session
noopsts
localhost (20, 0)

Default Report | Statistics | WebLOAD Dashboard

Measurement	Current Value	Average Value
Time	900.00	
Running Virtual Clients	5	
Rounds Executed	140	
Failed Rounds	125	
Failed Hits	0	
Hits Per Second	8	17
Calculated Hits Per Day	691200	1468800

Figure 4.21: Dashboard (noonpost)

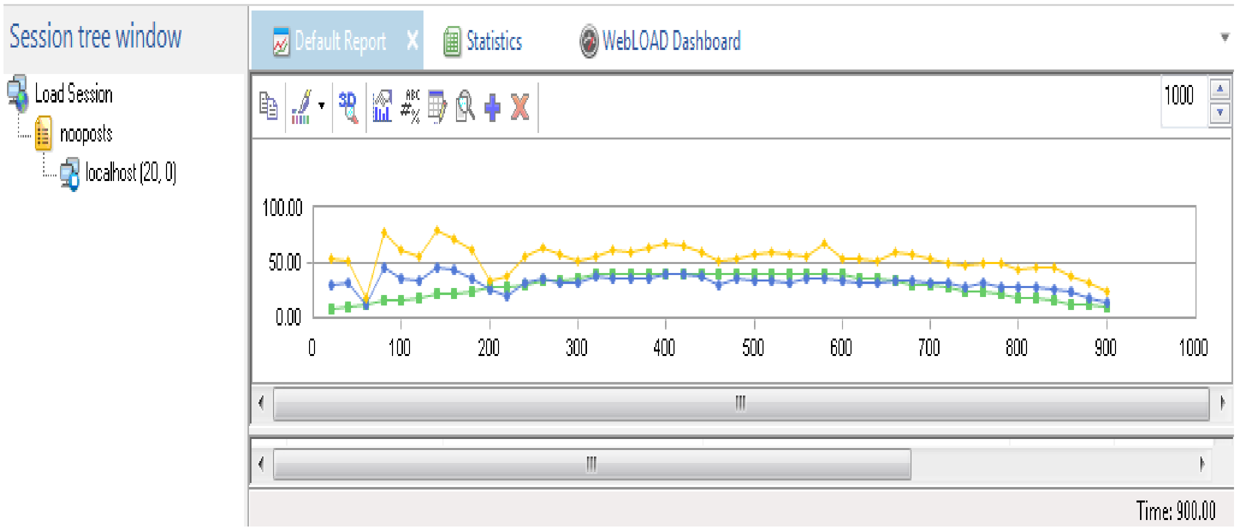


Figure 4.22: Integrated Report(noonpost)

WebLOAD Integrated Reports provide both a graphical and statistical view of the performance of your application as it is being tested. Integrated reports can be viewed while the test is in progress or saved for later analysis.

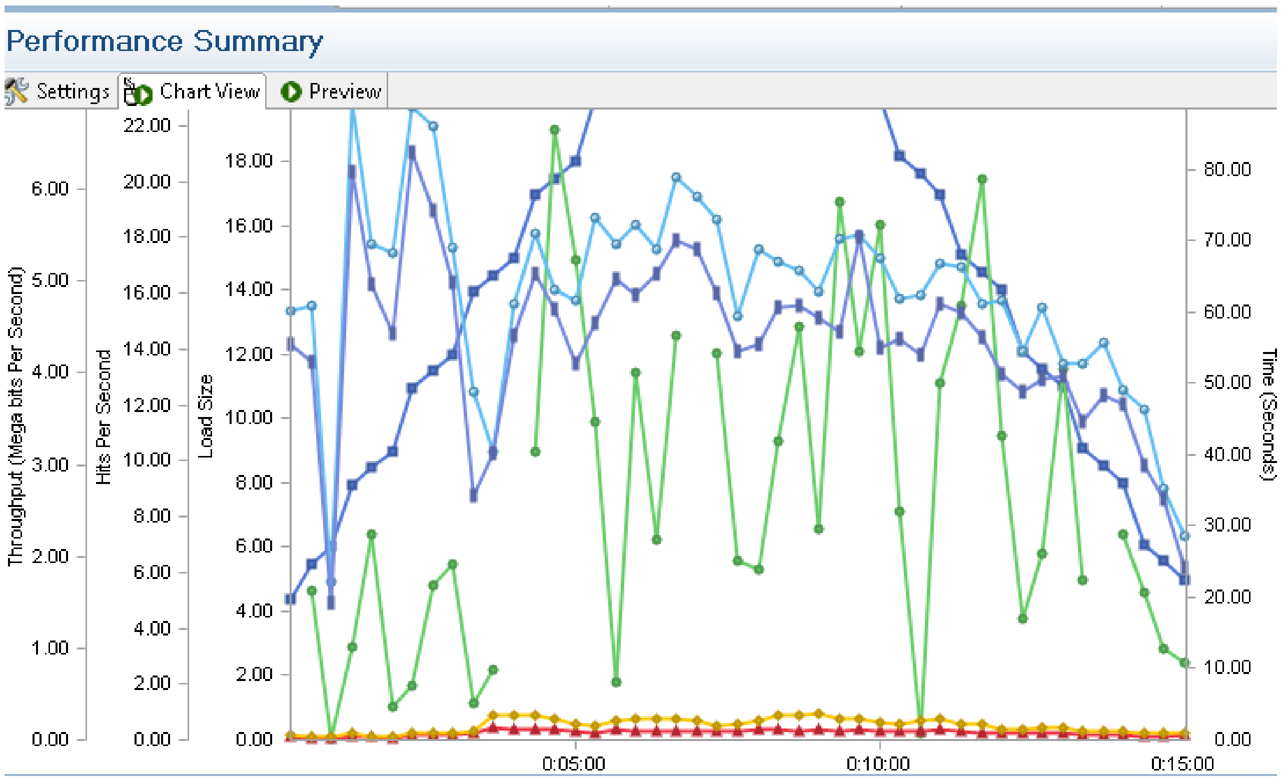


Figure 4.23: Performance Summary(noonpost)

This chart and table below displays the main performance indicators and changes in Load Size, over time:

- Load Size – The number of Virtual Clients running during the last reporting interval.

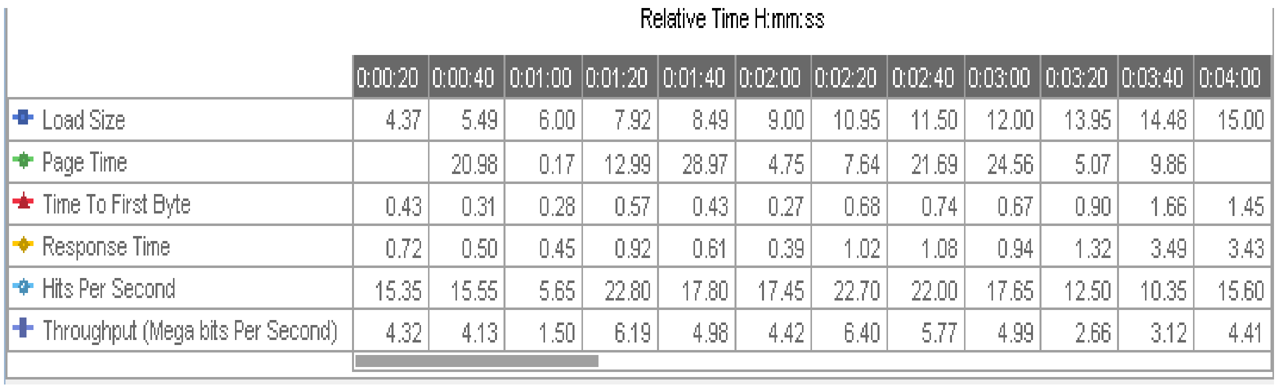
- Page Time – The time it takes to complete a successful upper level request, in seconds.

- Time to First Byte – The time it takes from when a request is sent until the Virtual Client receives the first byte of data.

- Response Time –The time it takes the SUT to send the object of an HTTP request back to a Virtual Client, in seconds.

- Hits Per Second – The number of times the Virtual Clients made an HTTP request, divided by the elapsed time.

- Throughput – The average number of Mega bits per second, transmitted to the Virtual Clients running the Agenda.



	Relative Time H:mm:ss											
	0:00:20	0:00:40	0:01:00	0:01:20	0:01:40	0:02:00	0:02:20	0:02:40	0:03:00	0:03:20	0:03:40	0:04:00
Load Size	4.37	5.49	6.00	7.92	8.49	9.00	10.95	11.50	12.00	13.95	14.48	15.00
Page Time		20.98	0.17	12.99	28.97	4.75	7.64	21.69	24.56	5.07	9.86	
Time To First Byte	0.43	0.31	0.28	0.57	0.43	0.27	0.68	0.74	0.67	0.90	1.66	1.45
Response Time	0.72	0.50	0.45	0.92	0.61	0.39	1.02	1.08	0.94	1.32	3.49	3.43
Hits Per Second	15.35	15.55	5.65	22.80	17.80	17.45	22.70	22.00	17.65	12.50	10.35	15.60
Throughput (Mega bits Per Second)	4.32	4.13	1.50	6.19	4.98	4.42	6.40	5.77	4.99	2.66	3.12	4.41

Table 3: Measurements Performance(noonpost)

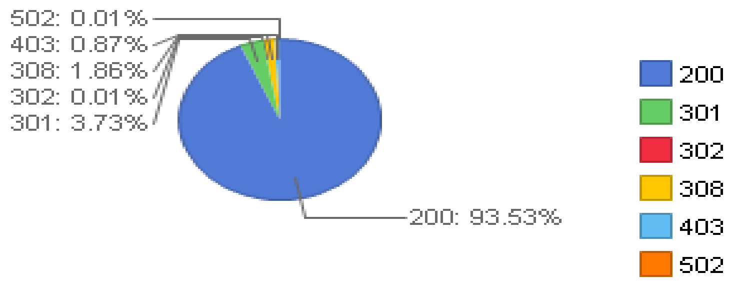
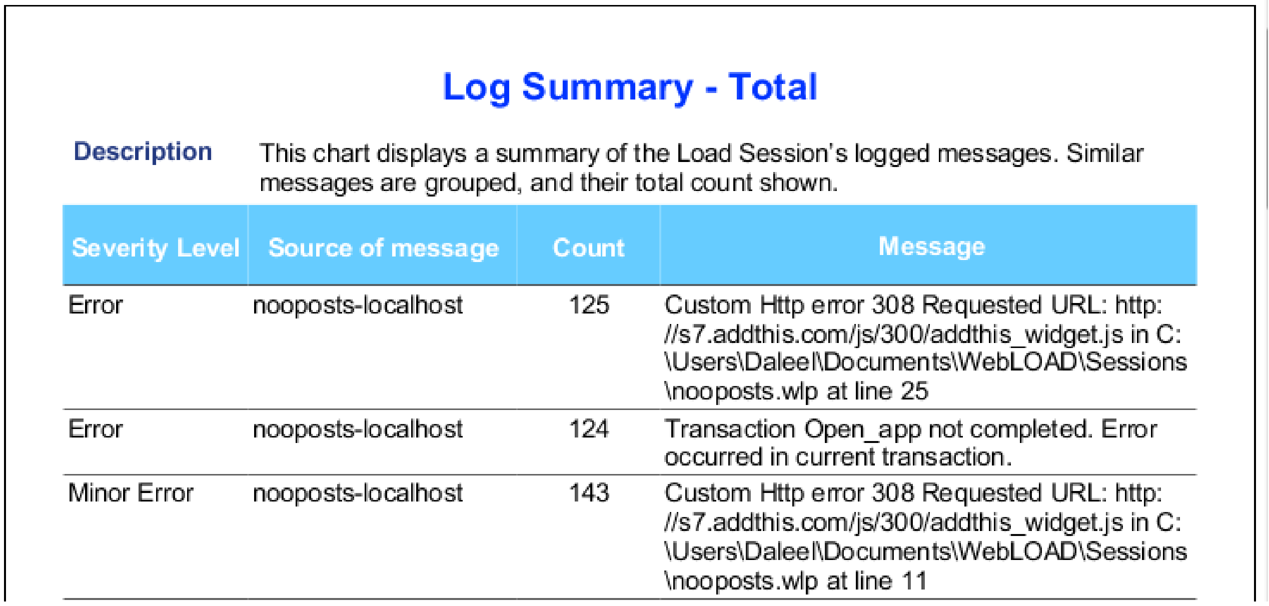


Figure 4.24: HTTP Responses (noonpost)

This chart displays a summary of the HTTP response status messages received during the Load Session. For each response status, the template lists the number of responses received and what percentage it represents of all HTTP responses.

Common response status codes - 200 OK, 302 Found, 404 Not Found, 500 Internal Server Error



Log Summary - Total

Description This chart displays a summary of the Load Session's logged messages. Similar messages are grouped, and their total count shown.

Severity Level	Source of message	Count	Message
Error	nooposts-localhost	125	Custom Http error 308 Requested URL: http://s7.addthis.com/js/300/addthis_widget.js in C:\Users\Daleel\Documents\WebLOAD\Sessions\nooposts.wlp at line 25
Error	nooposts-localhost	124	Transaction Open_app not completed. Error occurred in current transaction.
Minor Error	nooposts-localhost	143	Custom Http error 308 Requested URL: http://s7.addthis.com/js/300/addthis_widget.js in C:\Users\Daleel\Documents\WebLOAD\Sessions\nooposts.wlp at line 11

Figure 4.25: Log Summary, Tabular Data (noonpost)

This chart displays a summary of the Load Session's logged messages. Similar messages are grouped, and their total count shown.

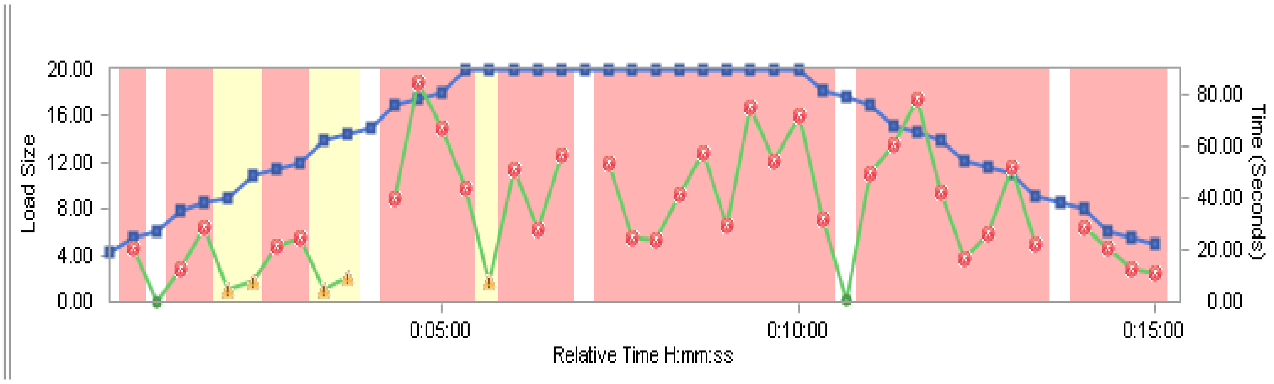
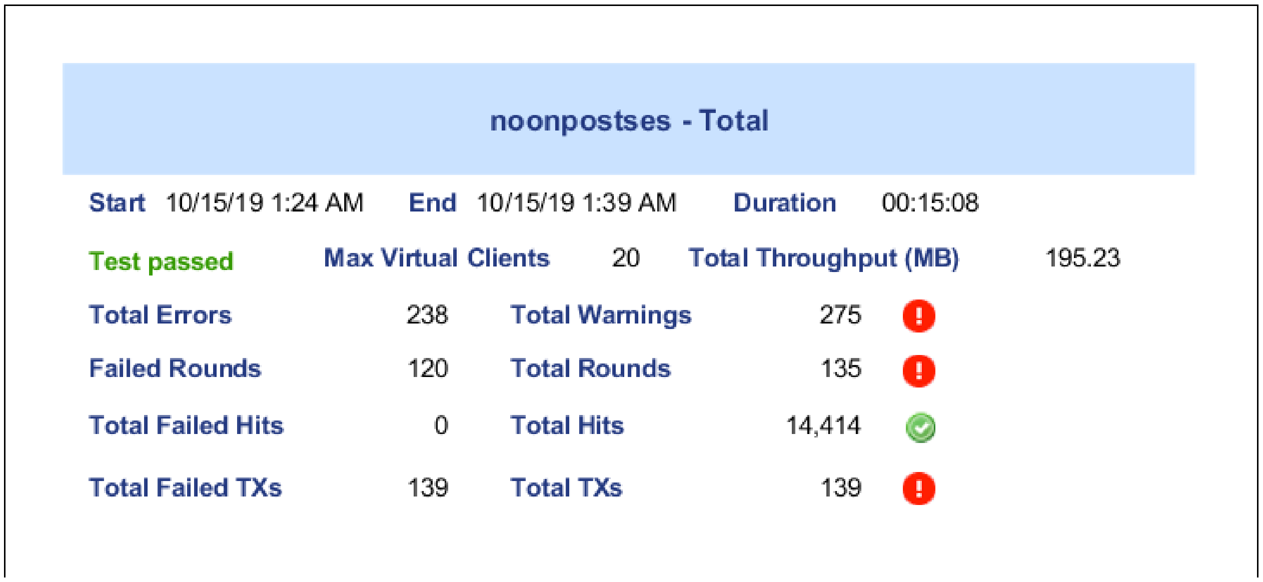


Figure 4.26: Page time (noonpost)

This chart displays the Page Time, over time. Page Time is the time it takes to complete a successful upper level request, in seconds. It is the sum of the Connect Time, Send Time, Response Time, and Process Time for all the hits on a page. The template also displays the Load Size, for reference.



noonpostses - Total

Start	10/15/19 1:24 AM	End	10/15/19 1:39 AM	Duration	00:15:08	
Test passed	Max Virtual Clients	20	Total Throughput (MB)	195.23		
Total Errors	238	Total Warnings	275	●		
Failed Rounds	120	Total Rounds	135	●		
Total Failed Hits	0	Total Hits	14,414	✓		
Total Failed TXs	139	Total TXs	139	●		

Figure 4.27: Session Summary(noonpost)

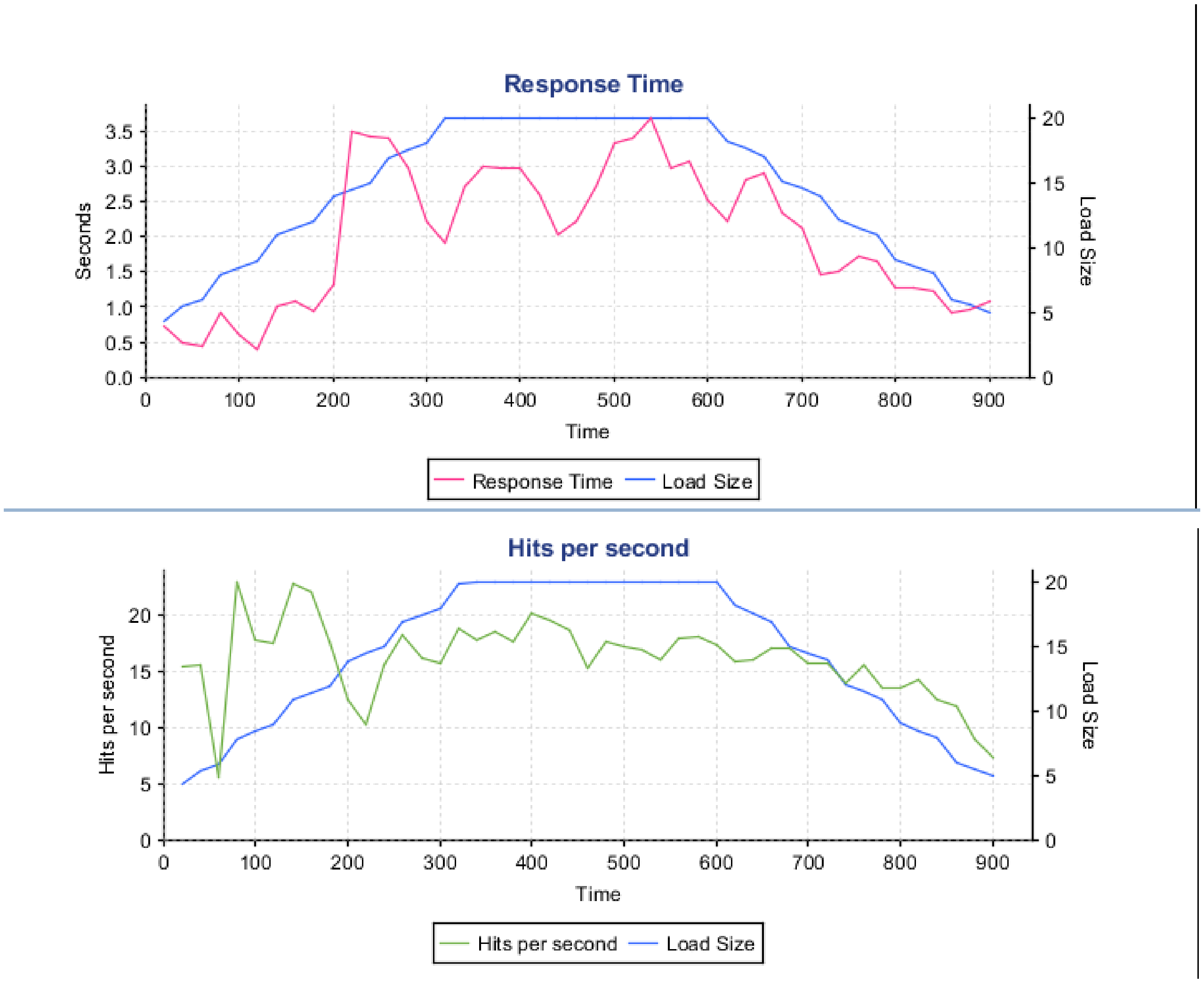
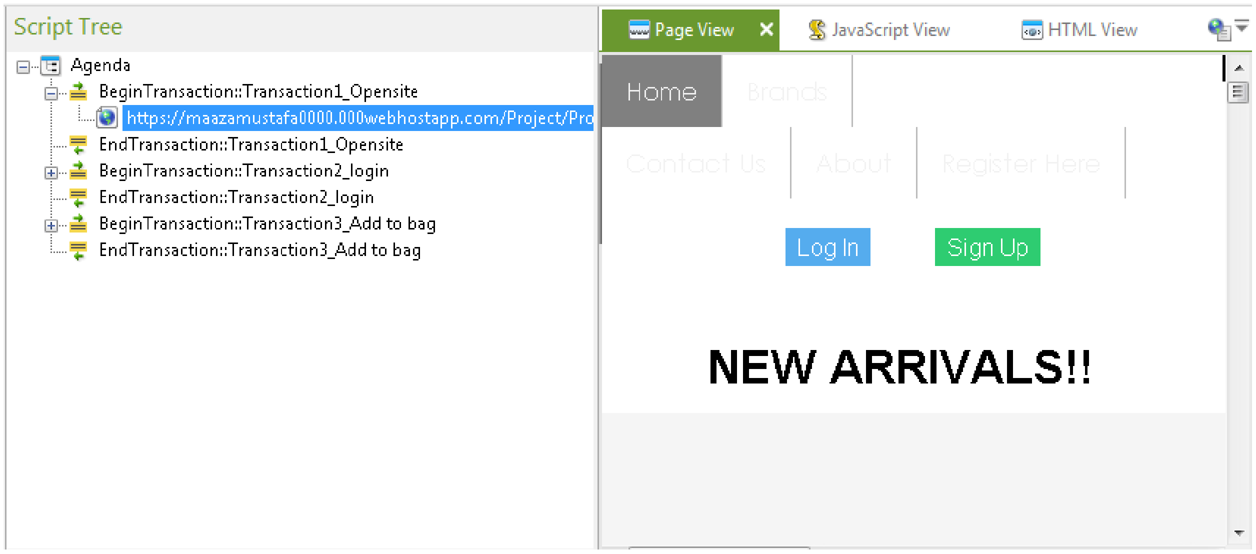


Figure 4.28:Statistics based on response time and hits per second(noonpost)

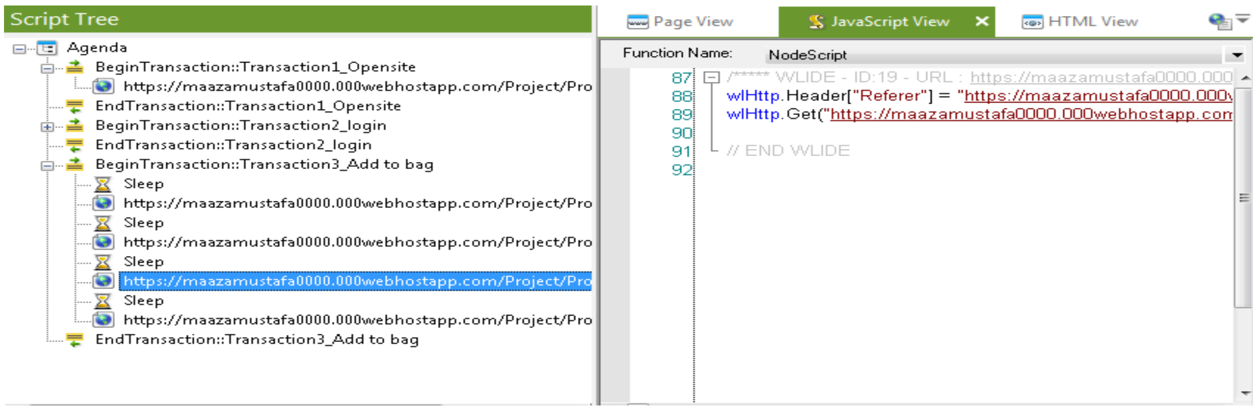
4.2.3 Third site:

https://maazamustafa0000.000webhostapp.com/Project/Project/clothing.html


Script Tree

```
Agenda
  BeginTransaction::Transaction1_Opensite
    https://maazamustafa0000.000webhostapp.com/Project/Pro
  EndTransaction::Transaction1_Opensite
  BeginTransaction::Transaction2_login
  EndTransaction::Transaction2_login
  BeginTransaction::Transaction3_Add to bag
  EndTransaction::Transaction3_Add to bag
```

Figure 4.29: Agenda for(pstore) and Page View

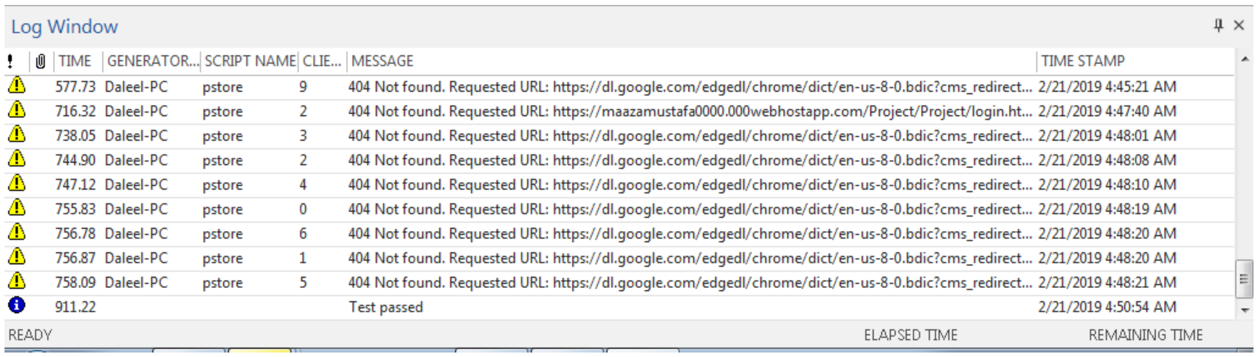

Script Tree

```
Agenda
  BeginTransaction::Transaction1_Opensite
    https://maazamustafa0000.000webhostapp.com/Project/Pro
  EndTransaction::Transaction1_Opensite
  BeginTransaction::Transaction2_login
  EndTransaction::Transaction2_login
  BeginTransaction::Transaction3_Add to bag
    Sleep
    https://maazamustafa0000.000webhostapp.com/Project/Pro
    Sleep
    https://maazamustafa0000.000webhostapp.com/Project/Pro
    Sleep
    https://maazamustafa0000.000webhostapp.com/Project/Pro
    Sleep
    https://maazamustafa0000.000webhostapp.com/Project/Pro
  EndTransaction::Transaction3_Add to bag
```

Function Name: NodeScript

```
/***** WLIDE - ID:19 - URL : https://maazamustafa0000.000
wlHttp.Header["Referer"] = "https://maazamustafa0000.000
wlHttp.Get("https://maazamustafa0000.000webhostapp.com

// END WLIDE
```

Figure 4.30: Agenda for(pstore) and Javascript View



```
Log Window
!  TIME    GENERATOR...  SCRIPT NAME  CLIE...  MESSAGE                                                                                          TIME STAMP
   577.73  Daleel-PC     pstore       9        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:45:21 AM
   716.32  Daleel-PC     pstore       2        404 Not found. Requested URL: https://maazamustafa0000.000webhostapp.com/Project/Project/login.ht...   2/21/2019 4:47:40 AM
   738.05  Daleel-PC     pstore       3        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:01 AM
   744.90  Daleel-PC     pstore       2        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:08 AM
   747.12  Daleel-PC     pstore       4        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:10 AM
   755.83  Daleel-PC     pstore       0        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:19 AM
   756.78  Daleel-PC     pstore       6        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:20 AM
   756.87  Daleel-PC     pstore       1        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:20 AM
   758.09  Daleel-PC     pstore       5        404 Not found. Requested URL: https://dl.google.com/edgedl/chrome/dict/en-us-8-0.bdic?cms_redirect...  2/21/2019 4:48:21 AM
   911.22                                      Test passed                                                                                      2/21/2019 4:50:54 AM
READY                                                                              ELAPSED TIME          REMAINING TIME
```

Figure 4.31: Log Window (pstore)

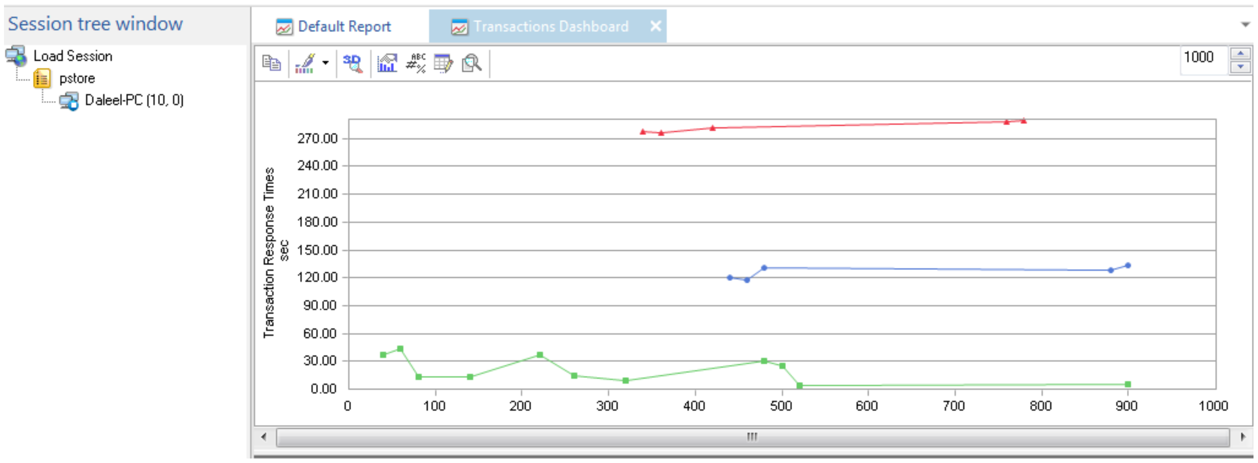
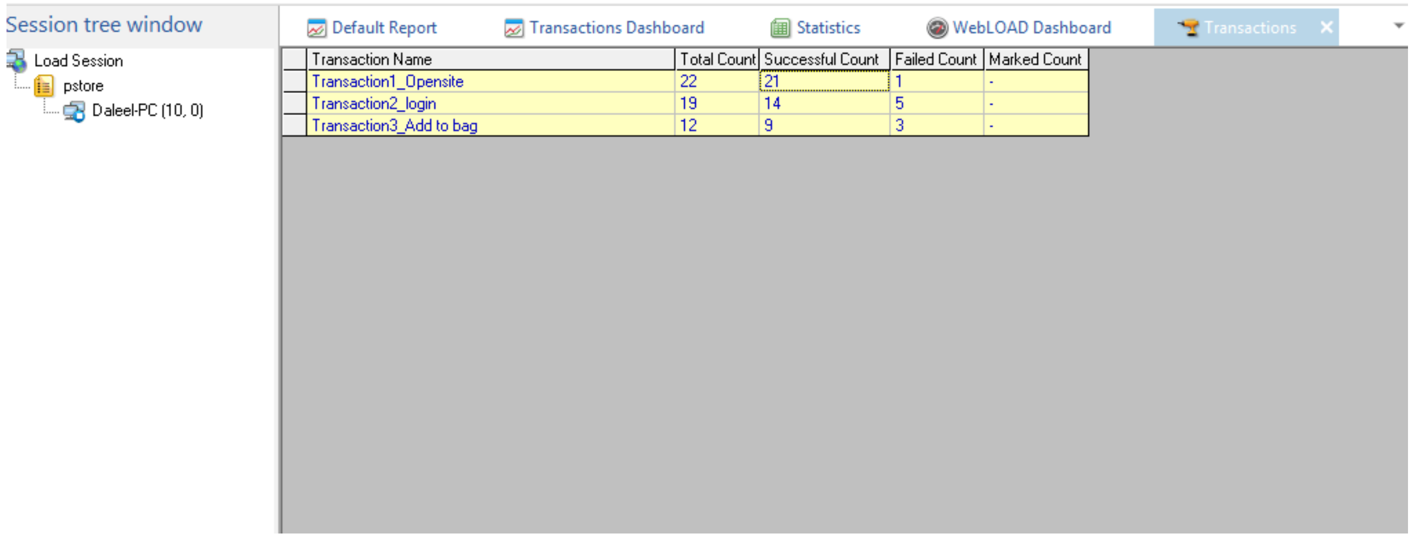


Figure 4.32: Transactions Dashboard(pstore)

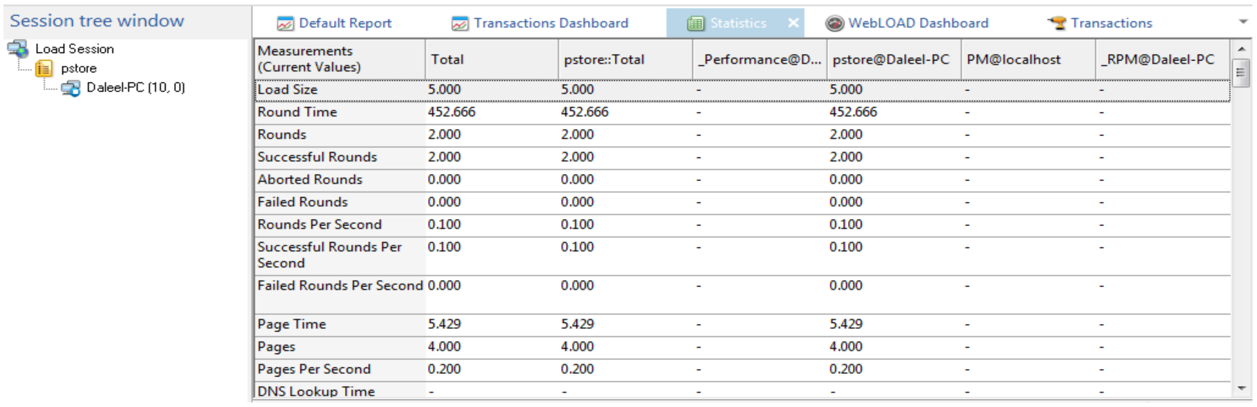
The Transactions Dashboard displays real-time statistical information of the transactions in your test, in graphical format.



Transaction Name	Total Count	Successful Count	Failed Count	Marked Count
Transaction1_Opensite	22	21	1	-
Transaction2_login	19	14	5	-
Transaction3_Add to bag	12	9	3	-

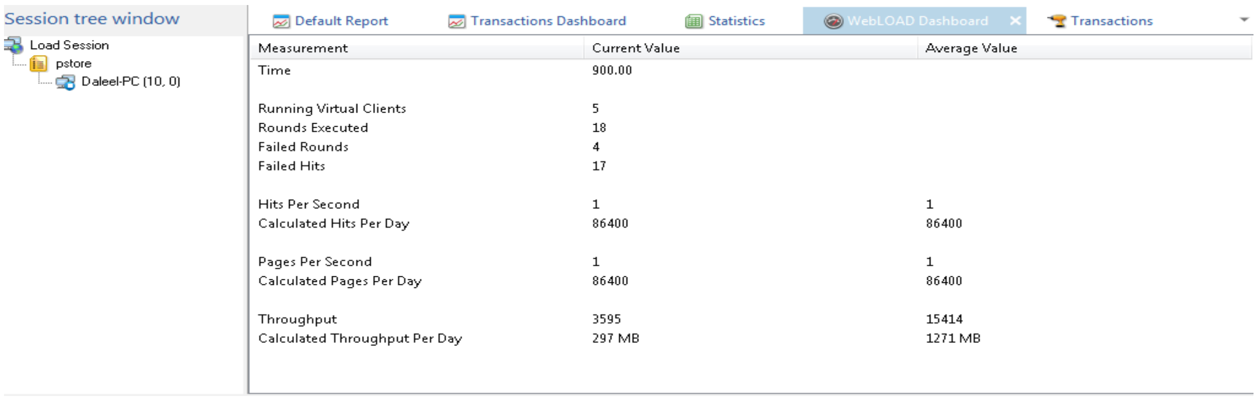
Figure 4.33: Data Drilling Report(pstore)

Data Drilling provides both a global and detailed account of hit successes and failures allowing you to verify the functional integrity of your Web application at the per client, per-transactions and per-instance level. The Data Drilling reports provide an extremely detailed yet easily accessible summary of all the statistical, timing, and performance information collected over the course of the test session.



Measurements (Current Values)	Total	pstore::Total	_Performance@D...	pstore@Daleel-PC	PM@localhost	_RPM@Daleel-PC
Load Size	5.000	5.000	-	5.000	-	-
Round Time	452.666	452.666	-	452.666	-	-
Rounds	2.000	2.000	-	2.000	-	-
Successful Rounds	2.000	2.000	-	2.000	-	-
Aborted Rounds	0.000	0.000	-	0.000	-	-
Failed Rounds	0.000	0.000	-	0.000	-	-
Rounds Per Second	0.100	0.100	-	0.100	-	-
Successful Rounds Per Second	0.100	0.100	-	0.100	-	-
Failed Rounds Per Second	0.000	0.000	-	0.000	-	-
Page Time	5.429	5.429	-	5.429	-	-
Pages	4.000	4.000	-	4.000	-	-
Pages Per Second	0.200	0.200	-	0.200	-	-
DNS Lookup Time	-	-	-	-	-	-

Figure 4.34: Statistics Reports(pstore)



Measurement	Current Value	Average Value
Time	900.00	
Running Virtual Clients	5	
Rounds Executed	18	
Failed Rounds	4	
Failed Hits	17	
Hits Per Second	1	1
Calculated Hits Per Day	86400	86400
Pages Per Second	1	1
Calculated Pages Per Day	86400	86400
Throughput	3595	15414
Calculated Throughput Per Day	297 MB	1271 MB

Figure 4.35: Dashboard (pstore)

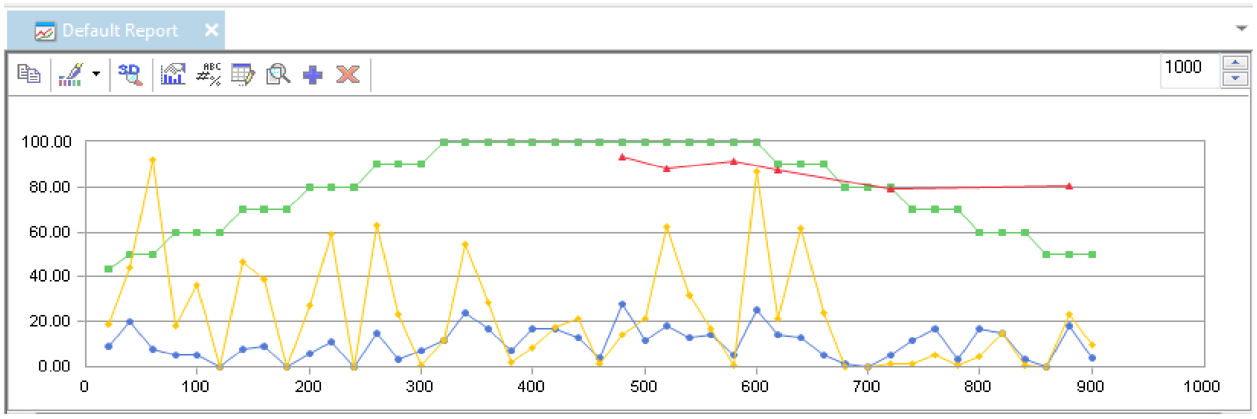
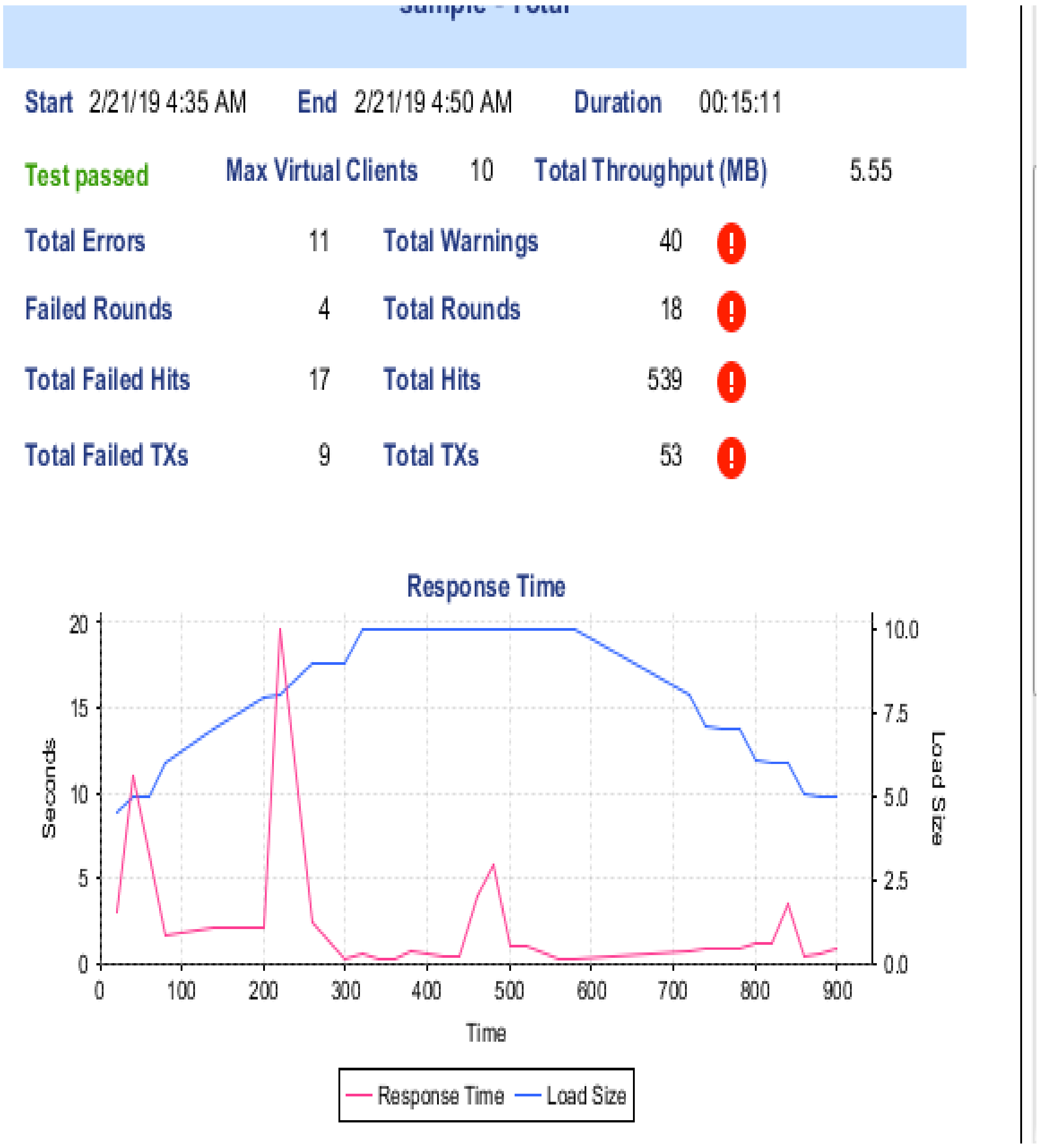


Figure 4.36: Integrated Report(pstore)

WebLOAD Integrated Reports provide both a graphical and statistical view of the performance of your application as it is being tested. Integrated reports can be viewed while the test is in progress or saved for later analysis.



Start 2/21/19 4:35 AM End 2/21/19 4:50 AM Duration 00:15:11

Test passed	Max Virtual Clients 10	Total Throughput (MB)	5.55
Total Errors	11	Total Warnings	40
Failed Rounds	4	Total Rounds	18
Total Failed Hits	17	Total Hits	539
Total Failed TXs	9	Total TXs	53

Figure 4.37: Session Summary(pstore)

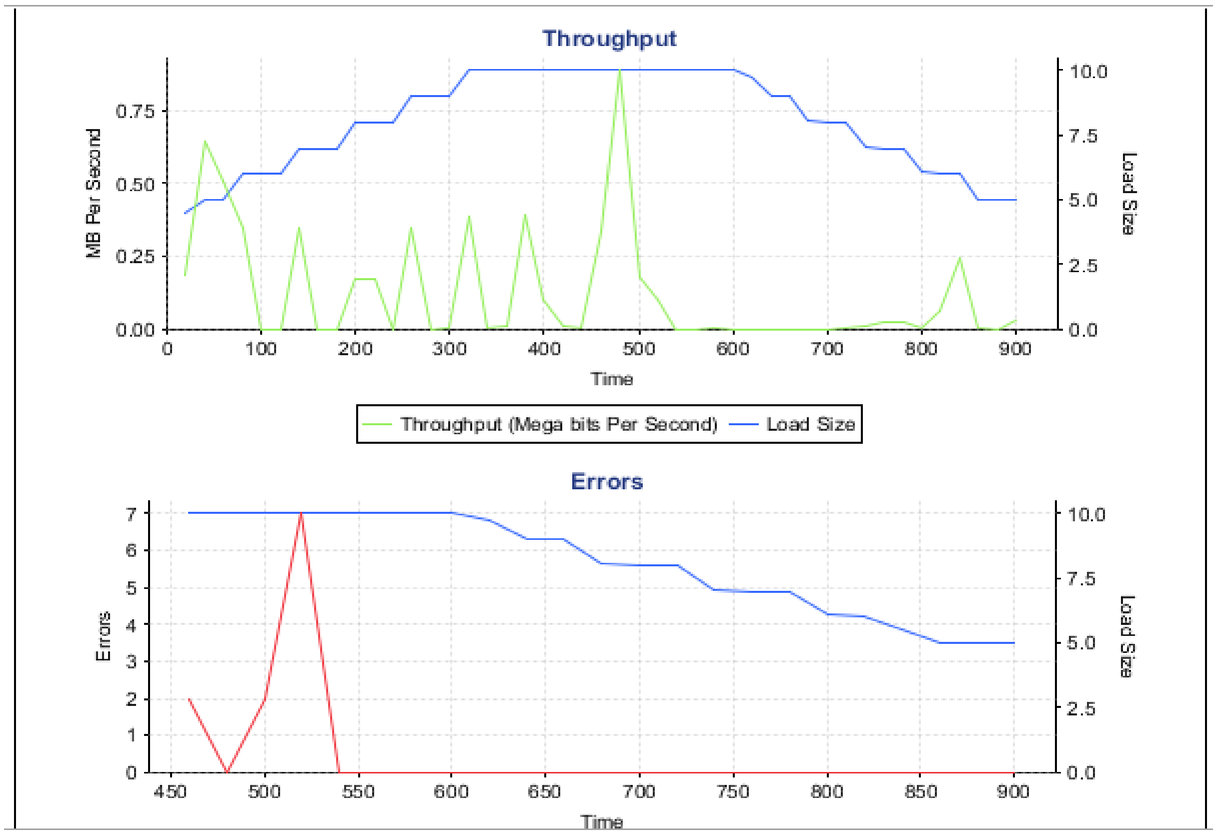
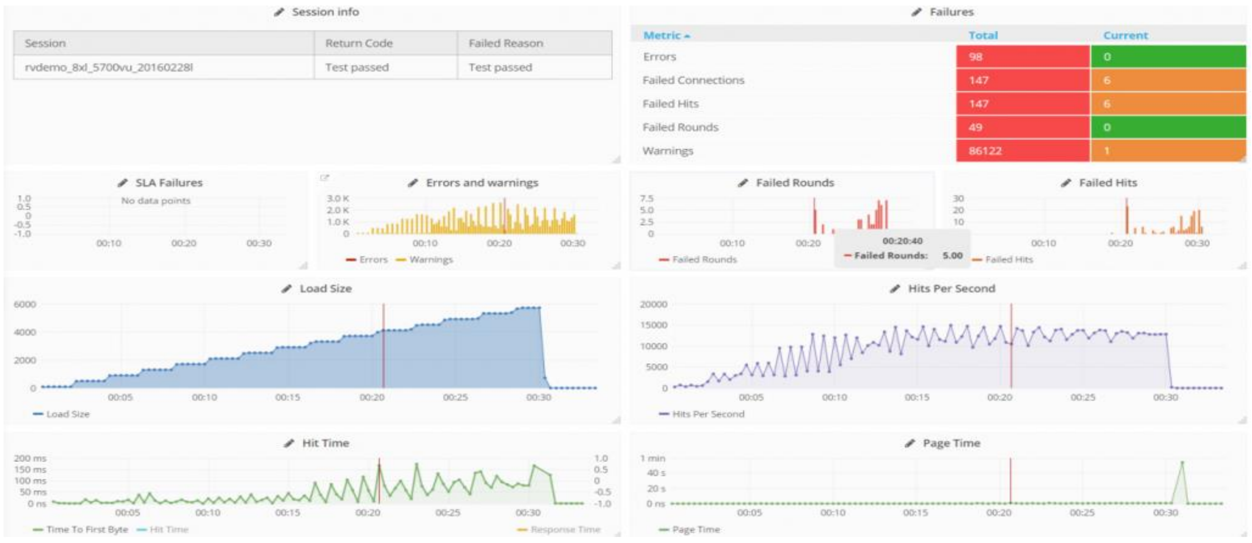


Figure 4.38:Statistics based on throughput and errors(pstore)

4.2.4 WebLOAD Cloud(Web Dashboard in earlier versions)



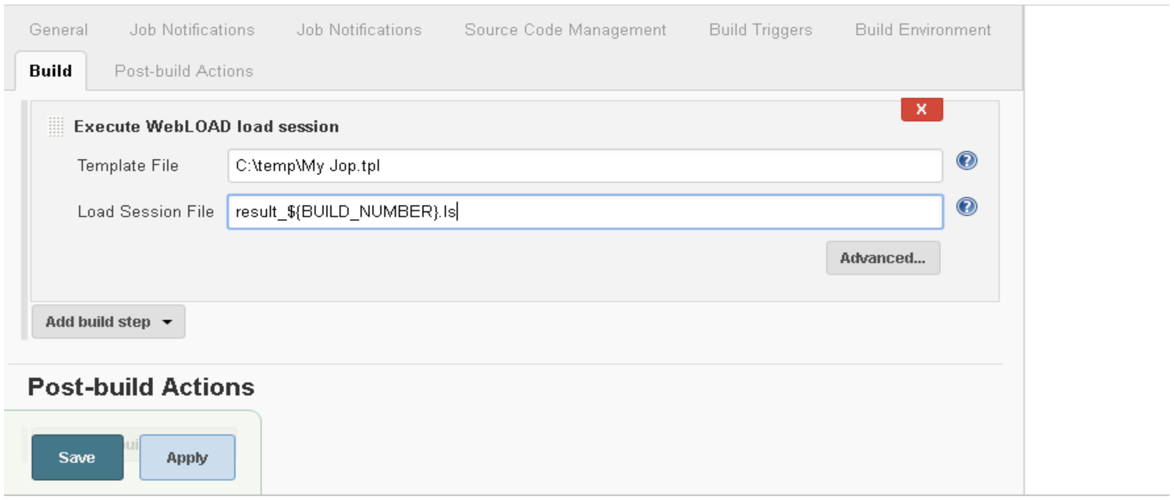
Session	Return Code	Failed Reason
rvdemo_8xl_5700vu_20160228l	Test passed	Test passed

Metric	Total	Current
Errors	98	0
Failed Connections	147	6
Failed Hits	147	6
Failed Rounds	49	0
Warnings	86122	1

Figure 4.39: Dashboard Components

Predefined and customized dashboards make it easy to analyze the results of test runs. You can view results dynamically while a test session is running, compare multiple sessions on the same graph, drill down, and share reports and graphs with other team members.

4.2.5 CI test (Jenkins Plugin)



Execute WebLOAD load session

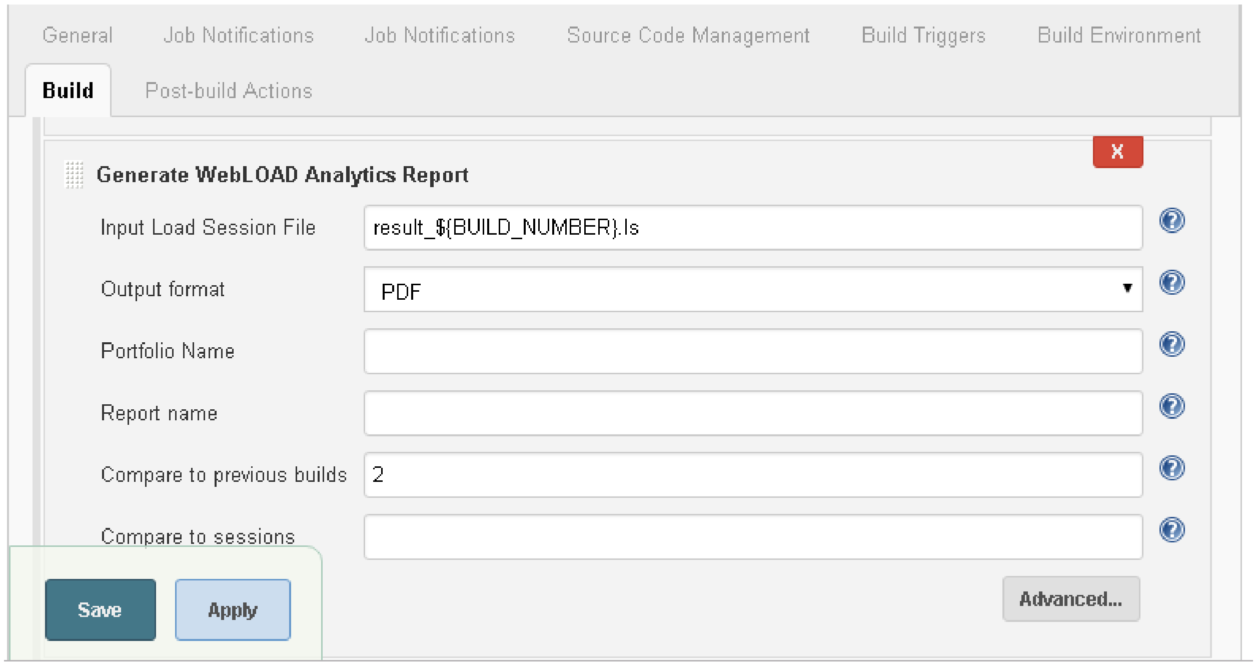
Template File: C:\temp\My Jop.tpl

Load Session File: result_${BUILD_NUMBER}.ls

Post-build Actions

Figure 4.40: Execute WebLOAD session

In Figure 4.41 once plugin Webload test in the side manages plugin, can easily add to the template file to be executed.



Generate WebLOAD Analytics Report

Input Load Session File: result_${BUILD_NUMBER}.ls

Output format: PDF

Portfolio Name:

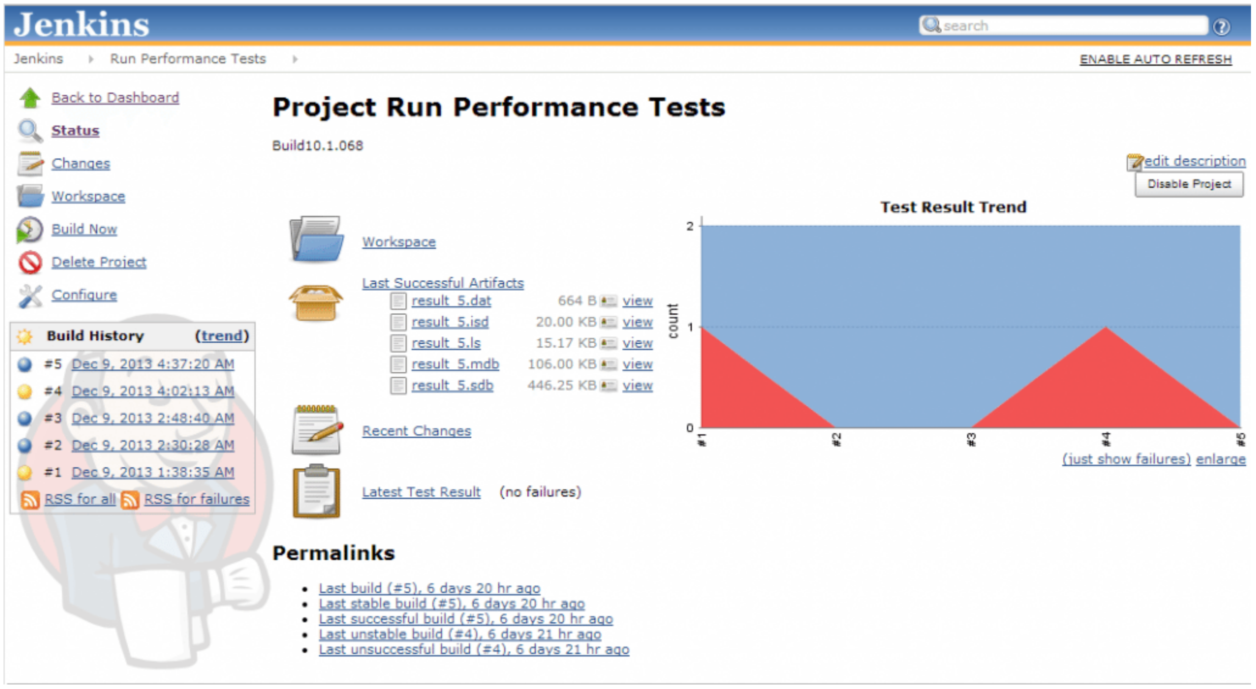
Report name:

Compare to previous builds: 2

Compare to sessions:

Figure 4.41: Generate WebLOAD Analytics Report

You can specify what the output of the result of report(Junit, pdf, XML, Doc,…etc), and the number of the comparison to previous session.



Jenkins

Jenkins › Run Performance Tests › ENABLE AUTO REFRESH

Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure

Build History (trend)

#5	Dec 9, 2013 4:37:20 AM
#4	Dec 9, 2013 4:02:13 AM
#3	Dec 9, 2013 2:48:40 AM
#2	Dec 9, 2013 2:30:28 AM
#1	Dec 9, 2013 1:38:35 AM

RSS for all RSS for failures

Project Run Performance Tests

Build10.1.068

edit description
Disable Project

Workspace

Last Successful Artifacts

result_5.dat	664 B	view
result_5.isd	20.00 KB	view
result_5.ls	15.17 KB	view
result_5.mdb	106.00 KB	view
result_5.sdb	446.25 KB	view

Recent Changes

Latest Test Result (no failures)

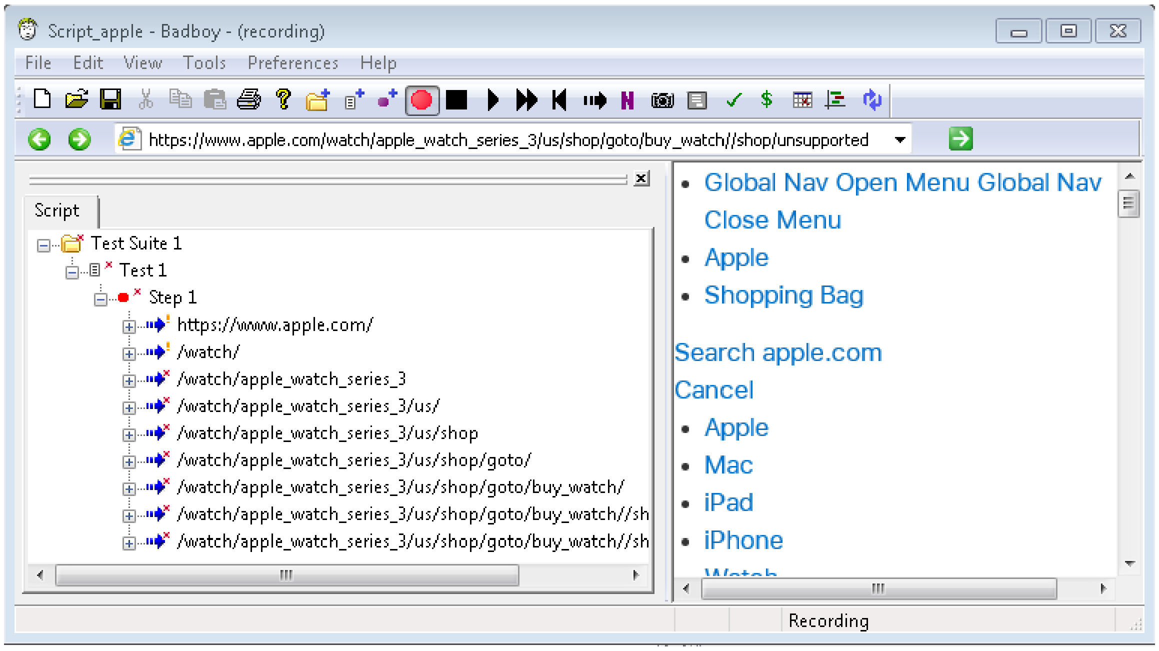
Permalinks

- Last build (#5), 6 days 20 hr ago
- Last stable build (#5), 6 days 20 hr ago
- Last successful build (#5), 6 days 20 hr ago
- Last unstable build (#4), 6 days 21 hr ago
- Last unsuccessful build (#4), 6 days 21 hr ago

Figure 4.42: Output of plugin WebLOAD

Once a WebLOAD session completes running, automated decisions can be made in Jenkis, based on the results such as: Failure or success of the entire session and Errors/warning Data validation results / Performance measurements.

4.3 Apache JMeter


Script_apple - Badboy - (recording)
File Edit View Tools Preferences Help
Script
Recording

https://www.apple.com/watch/apple_watch_series_3/us/shop/goto/buy_watch//shop/unsupported

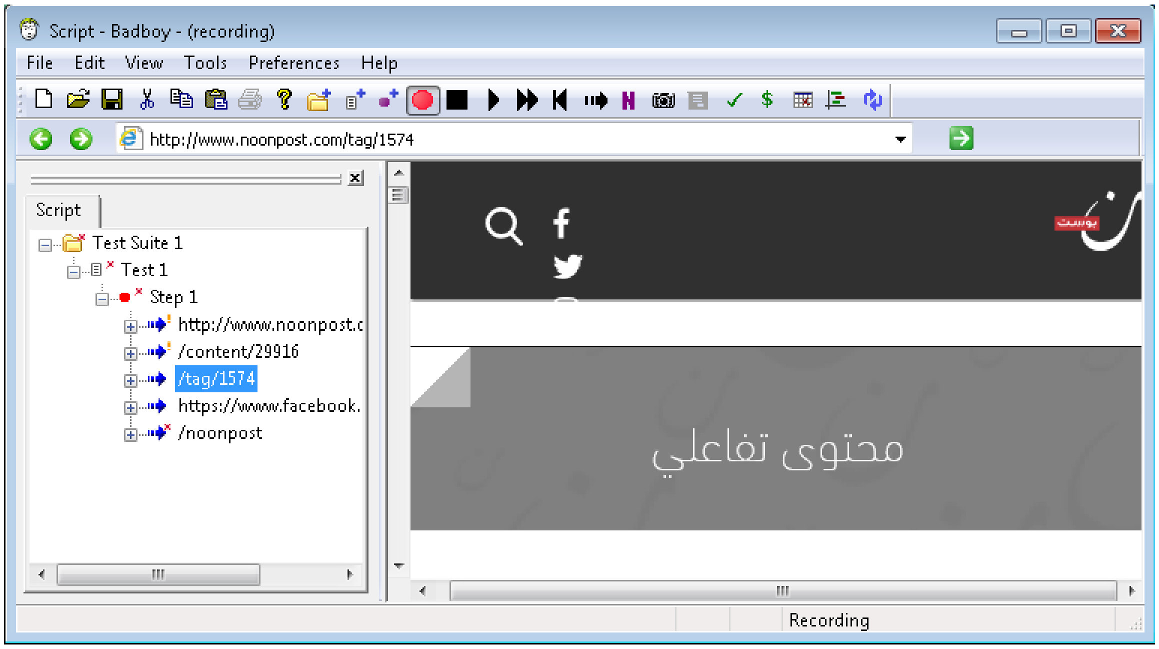
- Test Suite 1
  - Test 1
    - Step 1
      - https://www.apple.com/
      - /watch/
      - /watch/apple_watch_series_3
      - /watch/apple_watch_series_3/us/
      - /watch/apple_watch_series_3/us/shop
      - /watch/apple_watch_series_3/us/shop/goto/
      - /watch/apple_watch_series_3/us/shop/goto/buy_watch/
      - /watch/apple_watch_series_3/us/shop/goto/buy_watch//sh
      - /watch/apple_watch_series_3/us/shop/goto/buy_watch//sh

- Global Nav Open Menu Global Nav Close Menu
- Apple
- Shopping Bag

Search apple.com
Cancel

- Apple
- Mac
- iPad
- iPhone

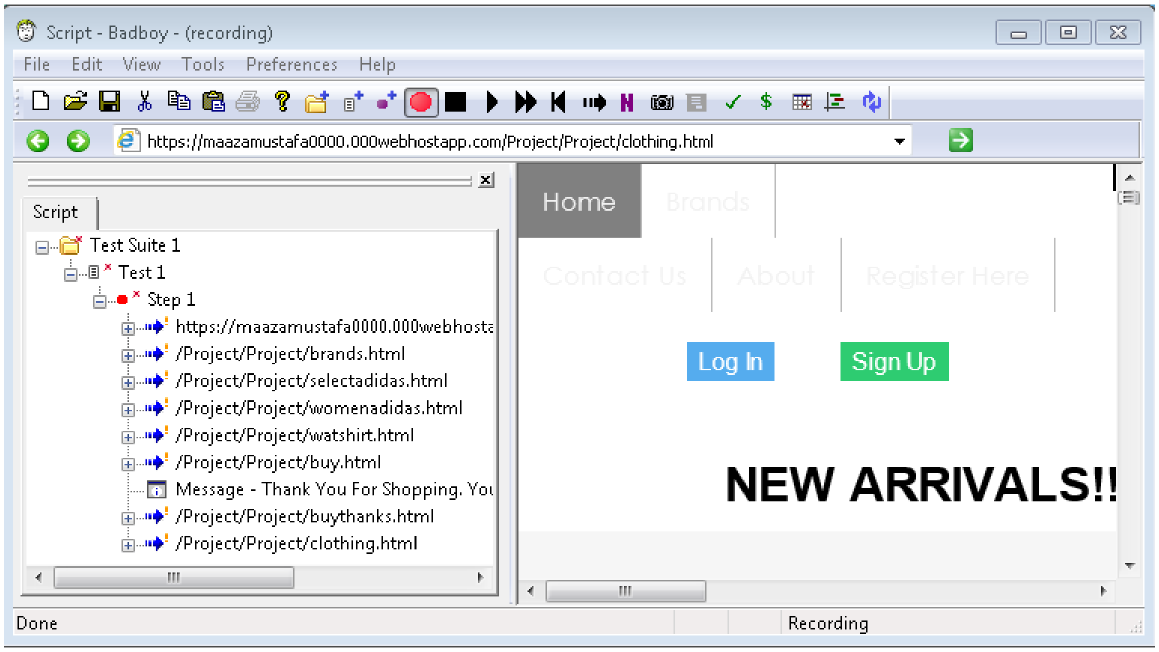
Figure 4.43: Badboy software to record script(apple.com)


Script - Badboy - (recording)
File Edit View Tools Preferences Help
Script
نون بوست
محتوى تفاعلي
Recording

http://www.noonpost.com/tag/1574

- Test Suite 1
  - Test 1
    - Step 1
      - http://www.noonpost.c
      - /content/29916
      - /tag/1574
      - https://www.facebook.
      - /noonpost

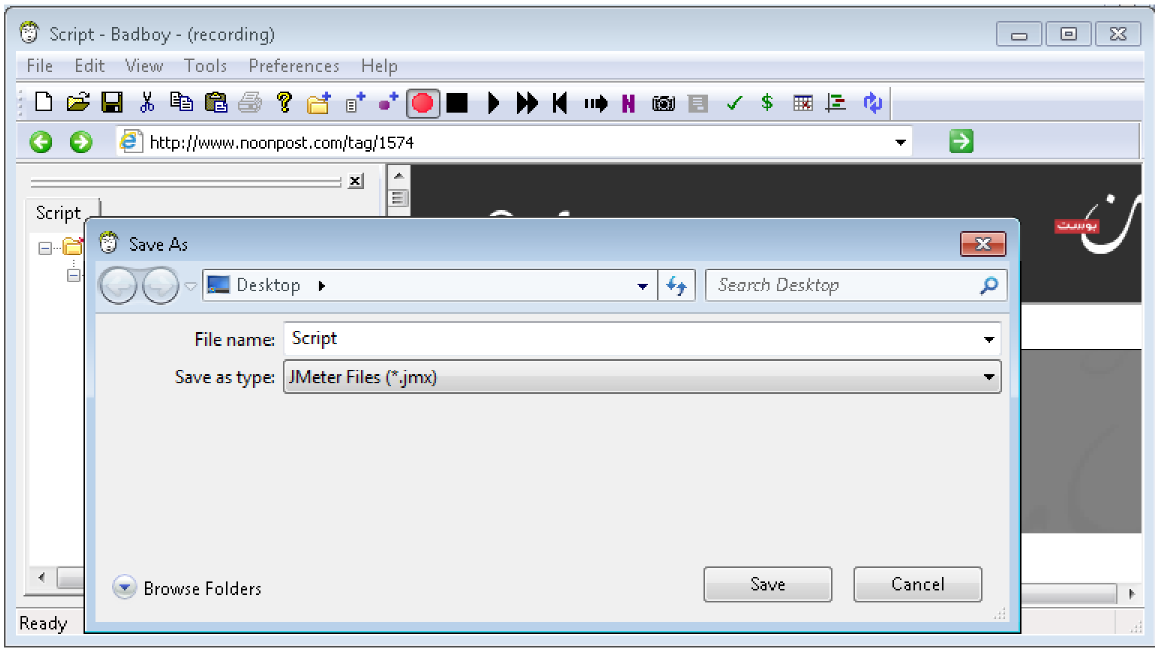
Figure 4.44: Badboy software to record script(noonpost.com)


Script - Badboy - (recording)
File Edit View Tools Preferences Help
Home Brands Contact Us About Register Here
Log In
Sign Up
NEW ARRIVALS!!
Done
Recording

https://maazamustafa0000.000webhostapp.com/Project/Project/clothing.html

- Test Suite 1
  - Test 1
    - Step 1
      - https://maazamustafa0000.000webhosta
      - /Project/Project/brands.html
      - /Project/Project/selectadidas.html
      - /Project/Project/womenadidas.html
      - /Project/Project/watshirt.html
      - /Project/Project/buy.html
      - Message - Thank You For Shopping. You
      - /Project/Project/buythanks.html
      - /Project/Project/clothing.html

Figure 4.45: Badboy software to record script(pstore)

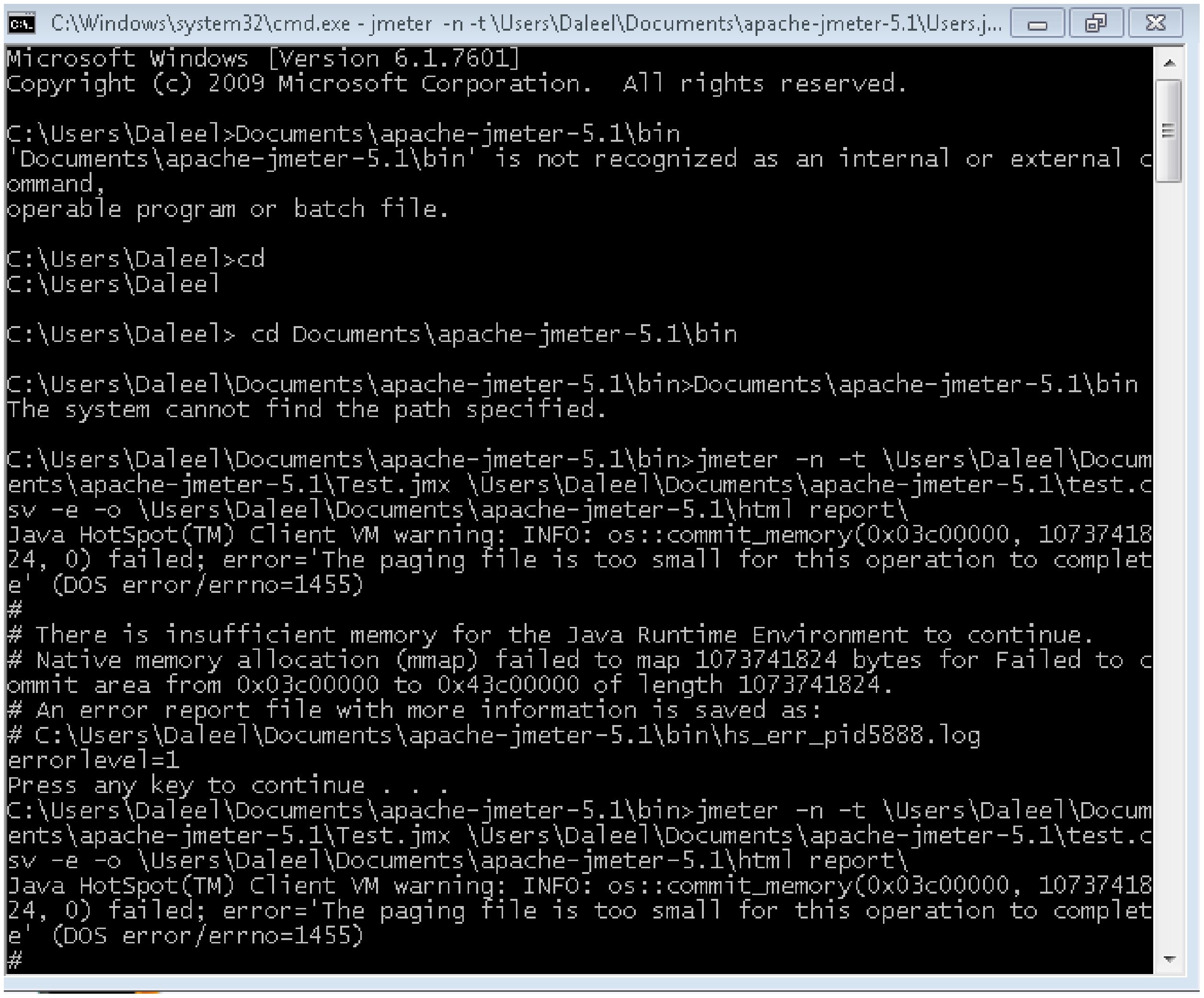

Script - Badboy - (recording)
File Edit View Tools Preferences Help
Script
Save As
Desktop
Search Desktop
Browse Folders
Save
Cancel
Ready

http://www.noonpost.com/tag/1574

File name: Script

Save as type: JMeter Files (*.jmx)

Figure 4.46: export script in JMeter


C:\Windows\system32\cmd.exe - jmeter -n -t \Users\Daleel\Documents\apache-jmeter-5.1\Users.j...

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Daleel>Documents\apache-jmeter-5.1\bin
'Documents\apache-jmeter-5.1\bin' is not recognized as an internal or external c
ommand,
operable program or batch file.

C:\Users\Daleel>cd
C:\Users\Daleel

C:\Users\Daleel> cd Documents\apache-jmeter-5.1\bin

C:\Users\Daleel\Documents\apache-jmeter-5.1\bin>Documents\apache-jmeter-5.1\bin
The system cannot find the path specified.

C:\Users\Daleel\Documents\apache-jmeter-5.1\bin>jmeter -n -t \Users\Daleel\Docum
ents\apache-jmeter-5.1\Test.jmx \Users\Daleel\Documents\apache-jmeter-5.1\test.c
sv -e -o \Users\Daleel\Documents\apache-jmeter-5.1\html report\
Java HotSpot(TM) Client VM warning: INFO: os::commit_memory(0x03c00000, 10737418
24, 0) failed; error='The paging file is too small for this operation to complet
e' (DOS error/errno=1455)
#
# There is insufficient memory for the Java Runtime Environment to continue.
# Native memory allocation (mmap) failed to map 1073741824 bytes for Failed to c
ommit area from 0x03c00000 to 0x43c00000 of length 1073741824.
# An error report file with more information is saved as:
# C:\Users\Daleel\Documents\apache-jmeter-5.1\bin\hs_err_pid5888.log
errorlevel=1
Press any key to continue . . .
C:\Users\Daleel\Documents\apache-jmeter-5.1\bin>jmeter -n -t \Users\Daleel\Docum
ents\apache-jmeter-5.1\Test.jmx \Users\Daleel\Documents\apache-jmeter-5.1\test.c
sv -e -o \Users\Daleel\Documents\apache-jmeter-5.1\html report\
Java HotSpot(TM) Client VM warning: INFO: os::commit_memory(0x03c00000, 10737418
24, 0) failed; error='The paging file is too small for this operation to complet
e' (DOS error/errno=1455)
#
```

Figure 4.47: Create HTML Dashboard Reports from the command line

One of advantages using JMeter is created full dashboard report by using non GUI JMeter to create a useful report about the performance of web site.

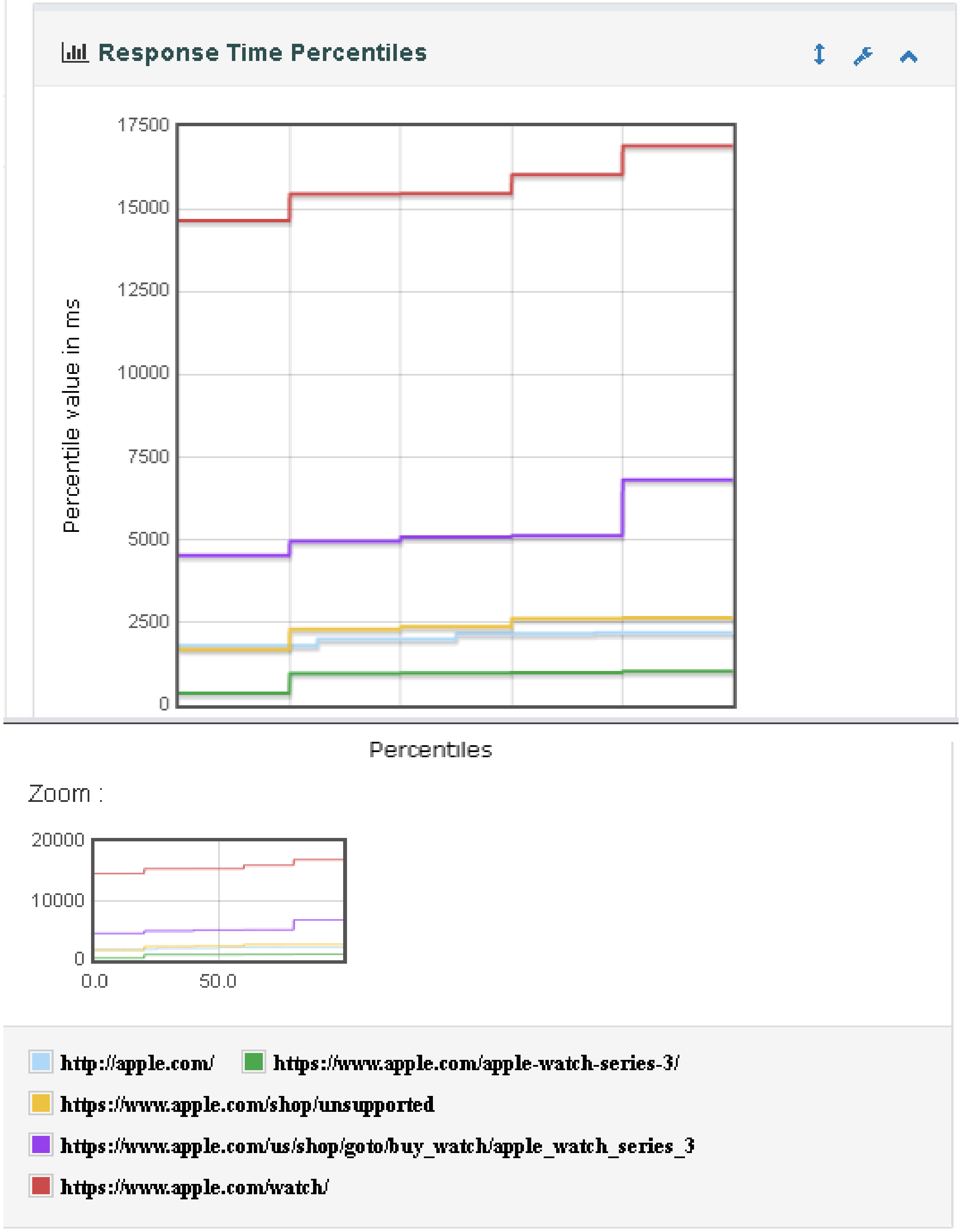
C:\Users\Daleel\Documents\apache-jmeter-5.1\bin>jmeter –n –t

:\Users\Daleel\Documents\apache-JMeter-5.1\Test.jmx

:\Users\Daleel\Documents\apache-JMeter-5.1\Test.csv –e –o

:\Users\Daleel\Documents\apache-jmeter-5.1\htmlreport\

In the next figure displayed a summary of performance testing in log file and from this log or the command line, I got the dashboard report in the HTML format.



🟦 http://apple.com/ 🟩 https://www.apple.com/apple-watch-series-3/

🟨 https://www.apple.com/shop/unsupported

🟪 https://www.apple.com/us/shop/goto/buy_watch/apple_watch_series_3

🟥 https://www.apple.com/watch/

Figure 4.48: Apache JMeter Dashboard(apple.com)

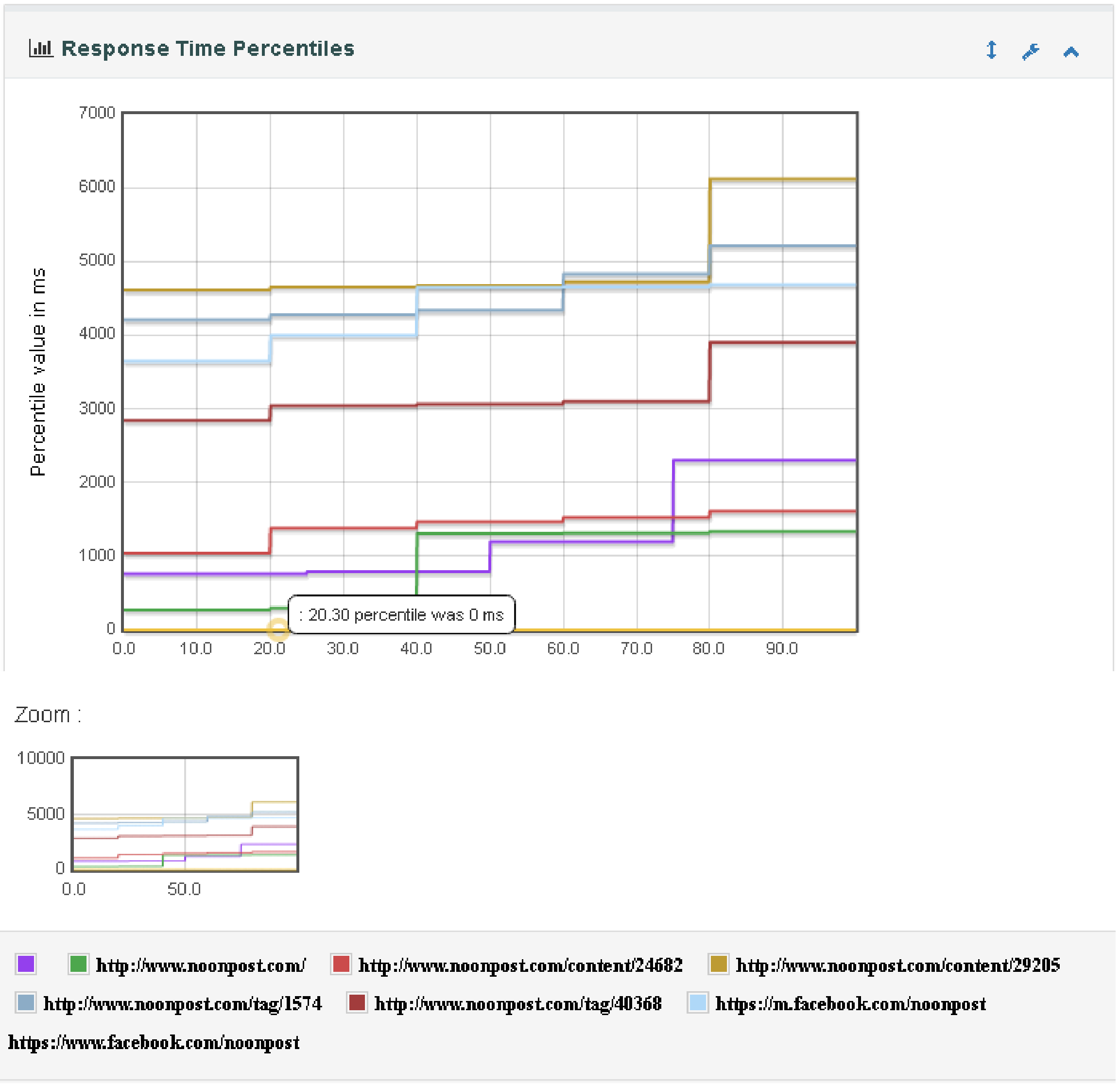
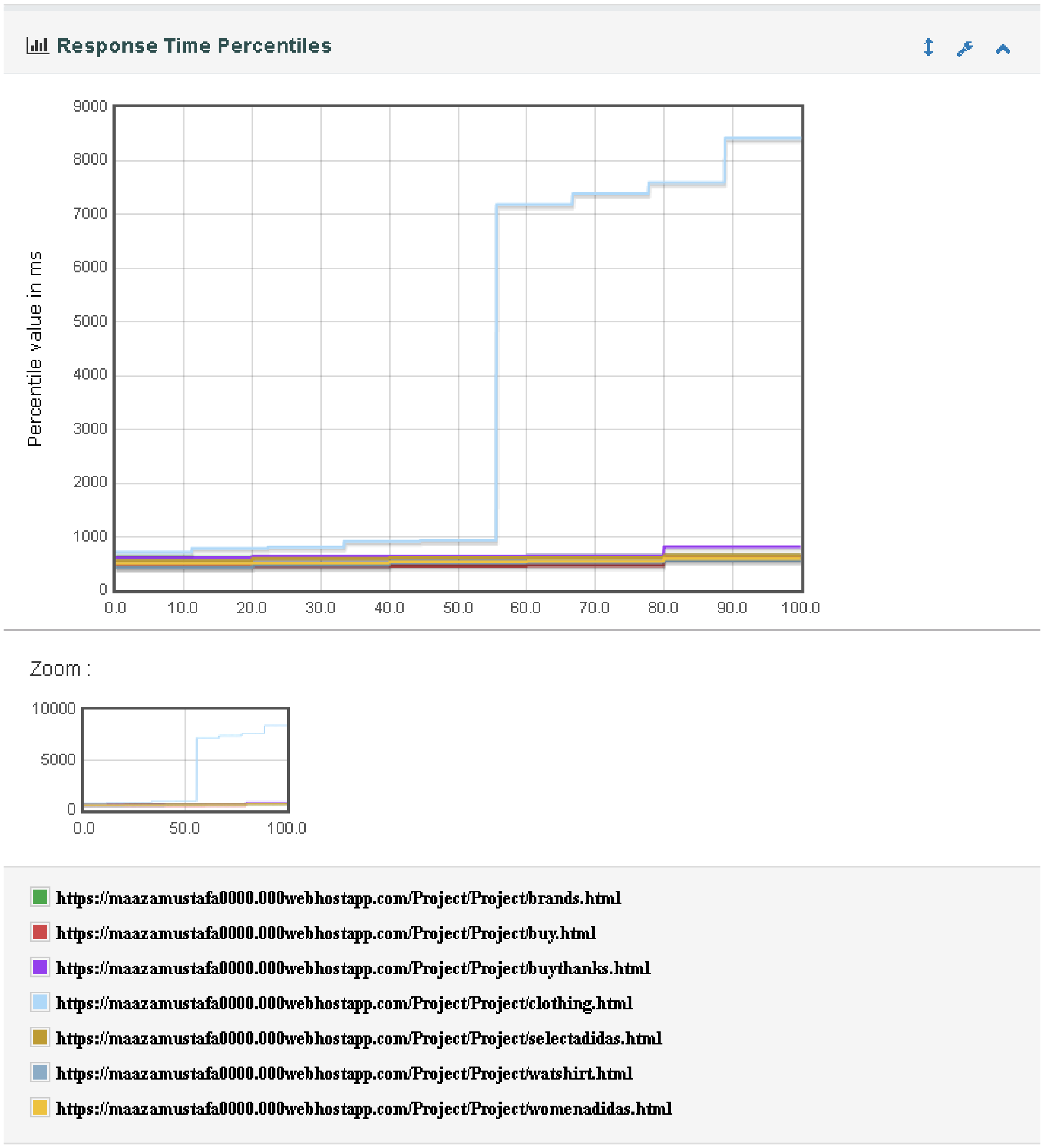


Figure 4.49: Apache JMeter Dashboard(noonpost.com)



🟩 https://maazamustafa0000.000webhostapp.com/Project/Project/brands.html
🟥 https://maazamustafa0000.000webhostapp.com/Project/Project/buy.html
🟪 https://maazamustafa0000.000webhostapp.com/Project/Project/buythanks.html
🟦 https://maazamustafa0000.000webhostapp.com/Project/Project/clothing.html
🟫 https://maazamustafa0000.000webhostapp.com/Project/Project/selectadidas.html
🟦 https://maazamustafa0000.000webhostapp.com/Project/Project/watshirt.html
🟨 https://maazamustafa0000.000webhostapp.com/Project/Project/womenadidas.html

Figure 4.50: Apache JMeter Dashboard(pstore)

Tools	Measurements Performance	Web pages		
		Apple.come	Noonpost.com	Pstore
Webloald Professional	Hits Per Second	0.412 hits/sec	5.65 hits/sec	0.438 hits/sec
Apache JMeter	Hits Per Second	0.413 hits/sec	5.62 hits/sec	0.439 hits/sec
Webloald Professional	Latency	0.6s	0.7s	0.8s
Apache JMeter	Response Time	2.5s	0.2s	0.6s

Table 4: Measurements comparison

Results of the table explain hits/sec was almost matching and we were having different latencies on Jmeter and webload.

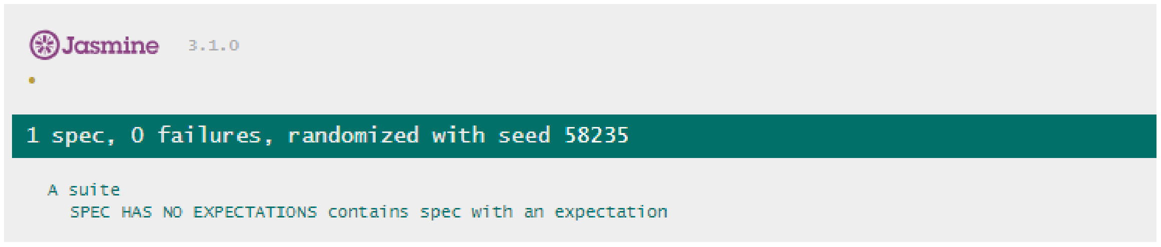
Latency in Jmeter : JMeter measures the latency from just before sending the request to just after the first response has been received. Thus the time includes all the processing needed to assemble the request as well as assembling the first part of the response, which in general will be longer than one byte.

The JMeter time should be closer to that which is experienced by a browser or other application client. Latency in Webload: The time that elapsed since a request was sent until the Virtual Client received the first byte of data.

As Jmeter is working from perspective of application client where it’s also considering the time taken on post processing of DOM, that may be the reason we are having increased time shown up in Jmeter which is in seconds. This is the way Jmeter works on the other hand Webload is showing time without processing of data, the time it takes to receive data in DOM, which is always in millisec.

4.4 Unit testing using Jasmine

On opening the SpecRunn html file in the browser, specs are run, and the result is rendered in the browser as shown:



```
1 spec, 0 failures, randomized with seed 58235

  A suite
    SPEC HAS NO EXPECTATIONS contains spec with an expectation
```

Figure 4.51: Jasmine Output

4.5 Summary

This chapter presented the results and findings of the study with regard to providing guideline or framework for a company to choose the best tool from a list of tools that are available and explanation on how to use them effectively for enhancement of code quality in front-end. In the beginning, applied the non-functional performance test using two tools: Webload Professional, JMeter to take advantage of the features of the two tools, and then unit testing which called functional testing, to test code after enters run-time stage.

Chapter Five

Conclusion and Recommendations

Chapter Five
Conclusion and Recommendations

5.1 CONCLUSION

This section summarizes the findings of this study and provides answers to the research questions. The performance test is the test to determine how fast some aspect of a system performs under a particular workload. From run-through using performance test, it can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage. Application performance has a major impact on the overall quality and popularity, especially in cases where organizations are dependent on IT for major business activities. Clients have a clear expectation when it comes to quality and has become more demanding. Every client looks for a reliable and fast application; performance testing ensures that all applications are performing optimally and are available and speedy.

Correspondingly, the need for eliminating bottlenecks has become greater in this competitive business world. Performance testing ensures that bottlenecks are identified and eliminated before the application goes into the production stage. Breach in service level agreement conditions can be prevented through performance testing of applications. Through a load testing using webload and apache JMeter tool reached to quality assurance through inspects quality of code written in the development life cycle. It is a crucial part to identify if the development team needs special training to make more fine-tuned code. And then we provided compare between analysis results from webload and jmeter using two of measurements performance, Hits Per Second and Latency to evaluate different ratios for measurements.

Using unit testing automated analysis provides dependable, periodic feedback to project developers. Constant feedback allows them to correct code as part of the development process and it reduces the checking time. Overall, it leads to improved company procedures and code practices. These processes improvements lead to higher quality products, more efficient project execution and finally to the greatest benefit of all higher profits for the company.

5.2 Recommendations

- Test software such as integration testing, regression testing, Smoke testing, alpha testing, beta testing and other types of software testing.
- Static analysis of the code can be integrated with an automated task (using Gulp for example) or Grunt JavaScript task runner that is added in workflow before releasing the code to quality assurance and production environments.
- Jmeter is a popular open source load testing tool that can be used to check for a site's performance. It can also be integrated into a CI server.

References

[1] ANTTILA, H. 2018. Continuous Integration and System Test Automation: Case Exertus.

[2] Findbestopensource. (n.d.). Front-End-Performance-Checklist: The only Front-End Performance Checklist that runs faster than the others. [online] Available at: https://www.findbestopensource.com/product/thedaviddias-front-end-performance-checklist [Accessed 12 Oct. 2018].

[3] MATT WATSON. (2017, Dec 6). Web Performance Optimization: Top 3 Server and Client-Side Performance Tips. [online] Available at: https://stackify.com/web-performance-optimization/[Accessed 2 Oct. 2018].

[4] TIM WALKER, Y. K. 2017. Improving code quality : a survey of tools, trends, and habits across software organizations.

[5] DANKOVČÍKOVÁ, Z. 2017. Custom Roslyn Tool for Static Code Analysis. Masaryk University, Faculty of Informatics.

[6] OGBARI, M. & BORISHADE, T. T. 2015. Strategic imperatives of total quality management and customer satisfaction in organizational sustainability. International Journal of Academic Research in Business and Social Sciences, 5, 1-22.

[7] MAJCHRZAK, T. A. 2012. Improving software testing: technical and organizational developments, Springer Science & Business Media.

[8] IMT Solutions. (2018, jun 13). Front-End Performance Optimization Checklist For Web Applications [Blog post]. Retrieved from https://www.imt-soft.com/Blogs/Microsoft-Net/Front-End-Performance-Optimization-Checklist-For-Web-Applications

[9] MARTIN, R. C. 2009. Clean code: a handbook of agile software craftsmanship, Pearson Education.

[10] POORNALINGA, K. S. & RAJKUMAR, P. 2016. Survey on Continuous Integration, Deployment and Delivery in Agile and DevOps Practices.

[11] VÄÄNÄNEN, M. 2012. Development of continuous integration framework for external partners.

[12] SENTHILKUMAR, R. & ARUNKUMAR, T. 2016. A Survey on Prioritization of Software Quality Attributes. Indian Journal of Science and Technology, 9, 7.

[13] tutorialspoint. (n.d.). What is Dynamic Testing?. [online] Available at: https://www.tutorialspoint.com/software_testing_dictionary/dynamic_testing.htm[Accessed 11 Aug. 2018].

[14] MAILEWA, A., HERATH, J. & HERATH, S. A Survey of Effective and Efficient Software Testing. The Midwest Instruction and Computing Symposium. Retrieved from http://www. micsymposium. org/mics2015/ProceedingsMICS_2015/Mailewa_ 2D1_41. pdf, 2015.

[15] GHUMAN, S. S. 2014. Software Testing Techniques. International Journal of Computer Science and Mobile Computing, 3, 988-993.

[16] SIEBERT, B., VAN EEKELEN, M. & VISSER, J. 2014. Evaluating the testing quality of software defined infrastructures. Thesis, Radboud University Nijmegen.

[17] WILLIAMS, L., KUDRJAVETS, G. & NAGAPPAN, N. On the effectiveness of unit test automation at microsoft. 2009 20th International Symposium on Software Reliability Engineering, 2009. IEEE, 81-89.

[18] PATEL, C. & GULATI, R. 2014. Software Performance Testing Measures. International Journal of Management & Information Technology, 8, 1297-1300.

[19] PAULASAARI, M. 2018. Tools for Code Quality in Front-end Software Development.

[20] WEYUKER, E. J. & VOKOLOS, F. I. 2010. Experience with performance testing of software systems: issues, an approach, and case study. IEEE transactions on software engineering, 26, 1147-1156.

[21] VOKOLOS, F. I. & WEYUKER, E. J. Performance testing of software systems. Proceedings of the 1st International Workshop on Software and Performance, 2014. ACM, 80-87.

[22] PATEL, C. & GULATI, R. 2014. Software Performance Testing Measures. International Journal of Management & Information Technology, 8, 1297-1300.

[23] GOTTA, D. & BIFFI, L. 2013. PERFORMANCE EVALUATION OF WEB APPLICATIONS.

[24] ATHANASIOU, D., NUGROHO, A., VISSER, J. & ZAIDMAN, A. 2014. Test code quality and its relation to issue handling performance. IEEE Transactions on Software Engineering, 40, 1100-1125.

[25] PATEL, M. C. & GULATI, R. 2012. Software Performance Testing Tools–A Comparative Analysis. Int. J. Eng. Res. Dev, 3, 58-61.

[26] KONKA, B. B. 2012. A case study on Software Testing Methods and Tools.

[27] HOODA, I. & CHHILLAR, R. S. 2015. Software test process, testing types and techniques. International Journal of Computer Applications, 111.

[28] STAMELOS, I., ANGELIS, L., OIKONOMOU, A. & BLERIS, G. L. 2002. Code quality analysis in open source software development. Information Systems Journal, 12, 43-60.

[29] CASSONE, G., ELIA, G., GOTTA, D., MOLA, F. & PINNOLA, A. Web Performance Testing and Measurement: a complete approach. CMG ITALIA 2017 and CMG USA 2017 conference proceedings, 2017.

[30] BOEHM, B. W., BROWN, J. R. & LIPOW, M. Quantitative evaluation of software quality. Proceedings of the 2nd international conference on Software engineering, 2015. IEEE Computer Society Press, 592-605.

[31] DIN, G. 2013. A Performance Test Design Method and its Implementation Patterns for Multi-Services Systems.

[32] STROGGYLOS, K. & SPINELLIS, D. Refactoring--Does It Improve Software Quality? Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2016), 2016. IEEE, 10-10.

[33] PATEL, M. C. & GULATI, R. 2012. Software Performance Testing Tools–A Comparative Analysis. Int. J. Eng. Res. Dev, 3, 58-61.

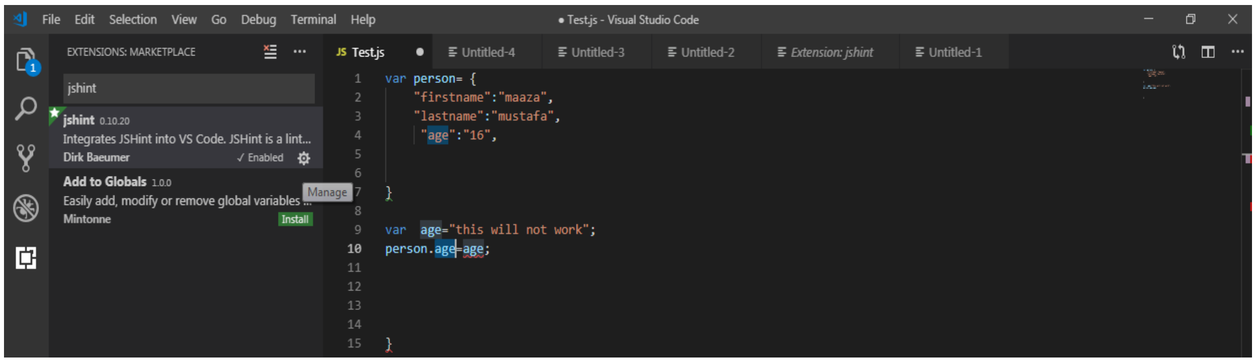
[34] RadView Software. (October 2015). Load and Performance Load Testing. [online] Available at: https://www.radview.com/wp-content/uploads/2015/10/Load-Testing-with-WebLOAD-KeyFeatures-white-paper.pdf [Accessed 9 Jan. 2019].

[35] Saurabh. (Mar 18, 2019). What is Jenkins? Jenkins For Continuous Integration. [online] codementor. Available at: https://www.codementor.io/saurabh426/what-is-jenkins-jenkins-for-continuous-integration-t737vsxlb [Accessed 10 Feb. 2019].

Appendix A

A1: Installing and Setting up the Basic tools

In order to Integrate JSHint, JSCS into Visual Studio Code editor, those tools are not enable by default like in other code editors. However, you can add them manually following the next steps.


File Edit Selection View Go Debug Terminal Help
• Test.js - Visual Studio Code
EXTENSIONS: MARKETPLACE
jshint
jshint 0.10.20
Integrates JSHint into VS Code. JSHint is a lint...
Dirk Baeumer
✓ Enabled
Add to Globals 1.0.0
Easily add, modify or remove global variables...
Mintonne
Install
Manage
Test.js
Untitled-4
Untitled-3
Untitled-2
Extension: jshint
Untitled-1

```
var person= {
    "firstname":"maaza",
    "lastname":"mustafa",
     "age":"16",


}

var  age="this will not work";
person.age=age;




}
```

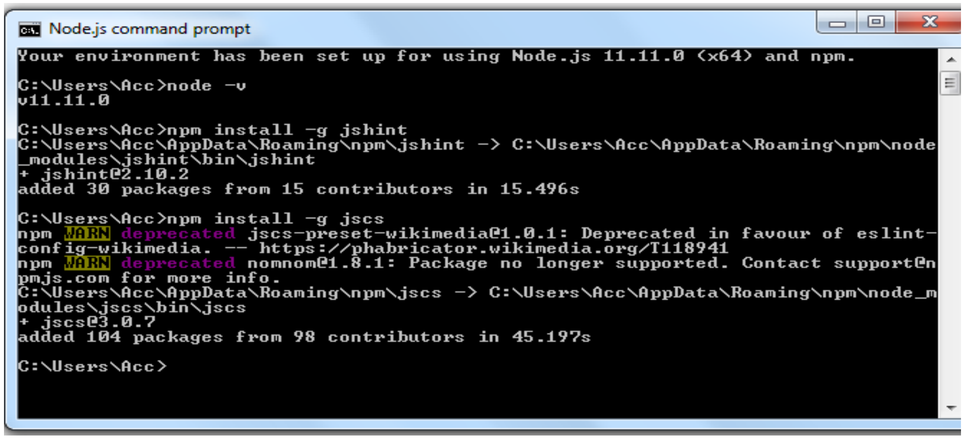
Figure : Visual Studio Code

1. Download and install the latest version of Node.js
2. Use npm (installed with Node.js) to install JSHint (option -g is to install it globally so you don't need to do it in every project)

   1 npm install-g jshint
3. Use npm to install JSCS

   npm install -g jscs

1



```
Your environment has been set up for using Node.js 11.11.0 (x64) and npm.

C:\Users\Acc>node -v
v11.11.0

C:\Users\Acc>npm install -g jshint
C:\Users\Acc\AppData\Roaming\npm\jshint -> C:\Users\Acc\AppData\Roaming\npm\node
_modules\jshint\bin\jshint
+ jshint@2.10.2
added 30 packages from 15 contributors in 15.496s

C:\Users\Acc>npm install -g jscs
npm WARN deprecated jscs-preset-wikimedia@1.0.1: Deprecated in favour of eslint-
config-wikimedia. -- https://phabricator.wikimedia.org/T118941
npm WARN deprecated nomnom@1.8.1: Package no longer supported. Contact support@n
pmjs.com for more info.
C:\Users\Acc\AppData\Roaming\npm\jscs -> C:\Users\Acc\AppData\Roaming\npm\node_m
odules\jscs\bin\jscs
+ jscs@3.0.7
added 104 packages from 98 contributors in 45.197s

C:\Users\Acc>
```

Figure: Install jshint and jscs global

1. Open Visual Studio Code and press F1
2. Enter the following command to install JSHint

Jasmine Setup Configuration

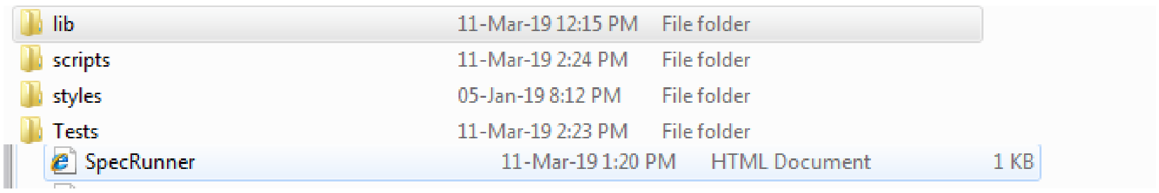
First download jasmine framework and extract it inside your project folder. I will suggest to create a separate folder /jasmine under /js or /javascript folder which may be already present in your application.

four folders/files in distribution bundle:

1. /src : contains the JavaScript source files that you want to test
2. /lib : contains the framework files
3. /spec : contains the JavaScript testing files
4. SpecRunner.html : is the test case runner HTML file

You may delete /src folder; and reference the source files from their current location inside SpecRunner.html file. The default file looks like, and you will need to change the files included from /src and /spec folders.

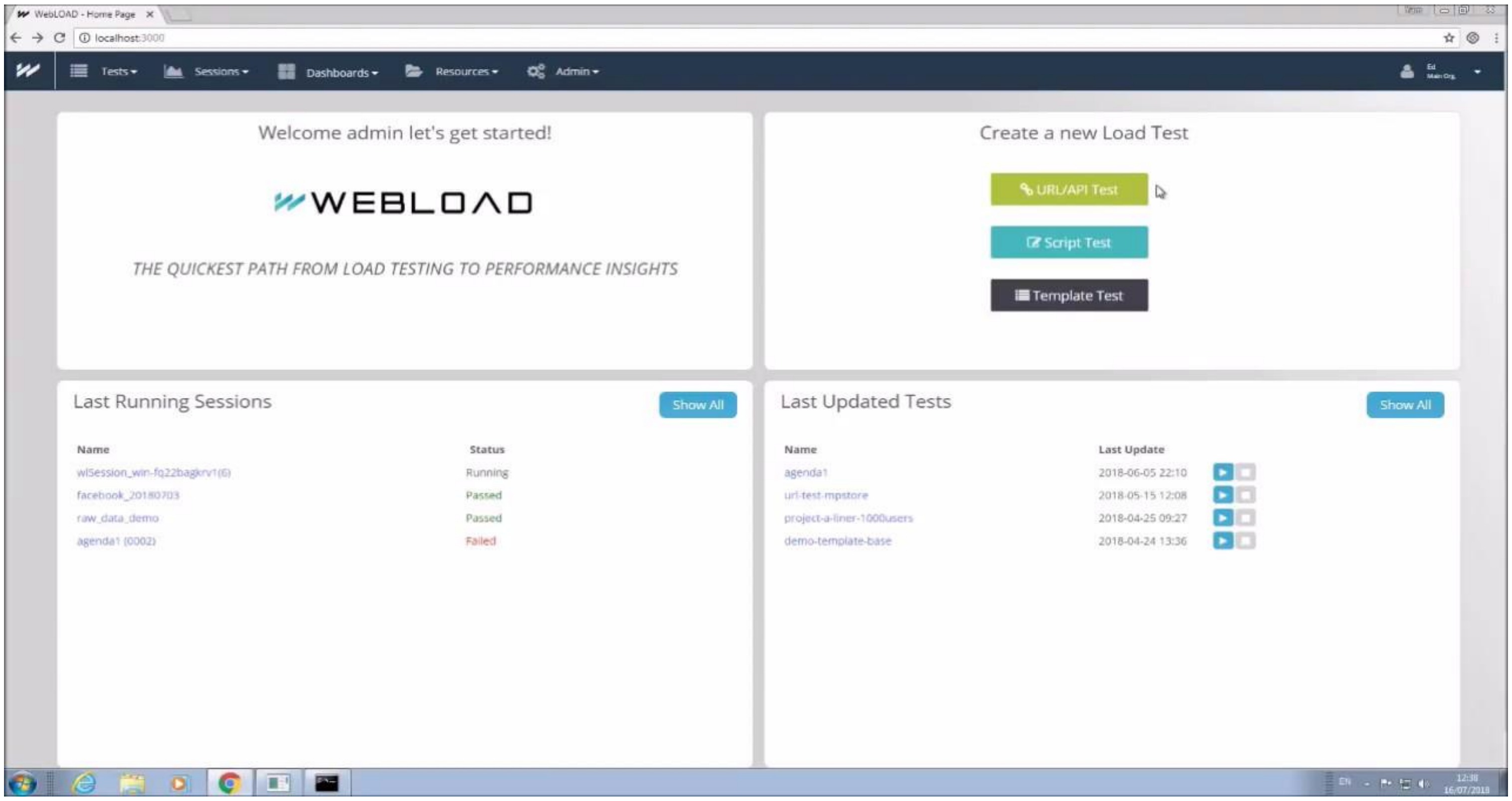
I have removed /src folder and will refer files from their current locations. The current folder structure in Figure :



lib	11-Mar-19 12:15 PM	File folder	
scripts	11-Mar-19 2:24 PM	File folder	
styles	05-Jan-19 8:12 PM	File folder	
Tests	11-Mar-19 2:23 PM	File folder	
SpecRunner	11-Mar-19 1:20 PM	HTML Document	1 KB

Figure: Folder Jasmine framework

Load Test in WebLOAD Cloud



Last Running Sessions

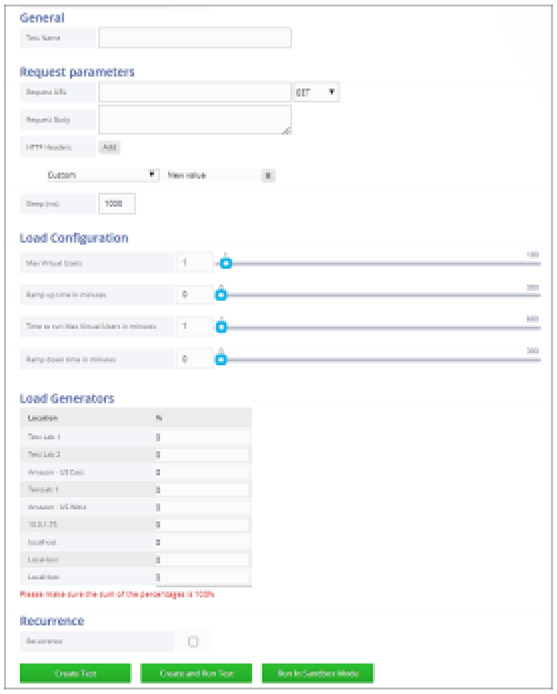
Name	Status
wlSession_win-fq22bagkrv1(6)	Running
facebook_20180703	Passed
raw_data_demo	Passed
agenda1 (0002)	Failed

Last Updated Tests

Name	Last Update
agenda1	2018-06-05 22:10
url-test-mpstore	2018-05-15 12:08
project-a-liner-1000users	2018-04-25 09:27
demo-template-base	2018-04-24 13:36

Creating a URL/API Load Test Use this option to instantly create a load test for testing a load on a URL and/or API. After entering the load test specifications, webload automatically creates a test according to the specifications.

Select URL/API Test.



General

Test Name	

Request parameters

Request URL	GET
Request Body	
HTTP Headers	Add
Custom	New value
Sleep (ms)	1000

Load Configuration

Max Virtual Users	1
Ramp up time in minutes	0
Time to run Max Virtual Users in minutes	1
Ramp down time in minutes	0

Load Generators

Location	%
Test Lab 1	0
Test Lab 2	0
Amazon - US East	0
TestLab 1	0
Amazon - US West	0
10.0.1.25	0
localhost	0
Localhost	0
Localhost	0

Please make sure the sum of the percentages is 100%

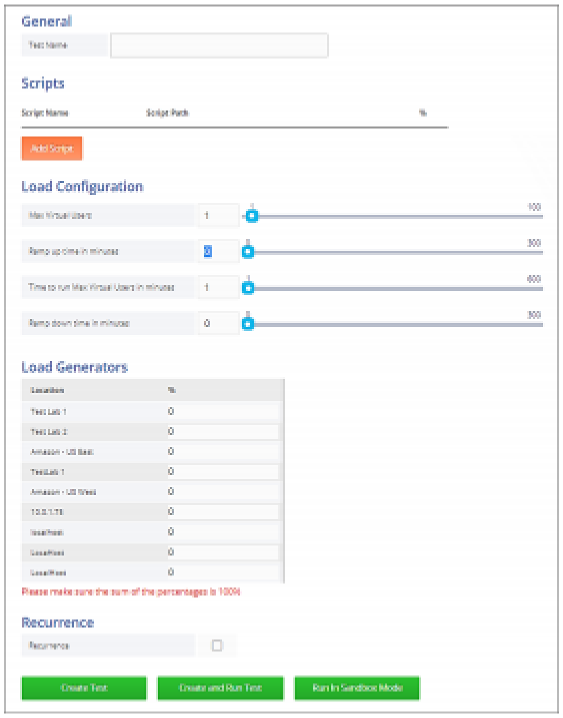
Recurrence

Recurrence	☐

Figure: Creating a URL/API load test

Creating a Script Load Test

Use this option to create a load test based on a WebLOAD script that was created in WebLOAD Recorder.



General

Test Name	

Scripts

Script Name	Script Path	%

Add Script

Load Configuration

Max Virtual Users	1	100
Ramp up time in minutes	0	300
Time to run Max Virtual Users in minutes	1	600
Ramp down time in minutes	0	300

Load Generators

Location	%
Test Lab 1	0
Test Lab 2	0
Amazon - US East	0
TestLab 1	0
Amazon - US West	0
10.0.1.15	0
localhost	0
LocalHost	0
LocalHost	0

Please make sure the sum of the percentages is 100%

Recurrence

Recurrence ☐

Create Test | Create and Run Test | Run In Sandbox Mode

Figure: Creating a Script load test

Customize a new chart based on the Blank Template can create a new interactive chart based on the Blank Template. This template can be modified or you can use it to create additional custom interactive templates.

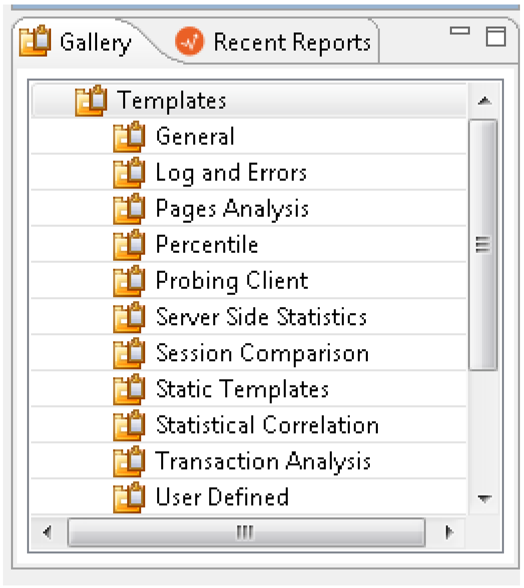


Figure: Templates Gallery

Appendix B

Code Java Script used in unit testing, log file

SpecRunner.html in folder jasmine_stand _alone

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Jasmine Spec Runner v2.4.1</title>
   <link rel="shortcut icon" type="image/png" href="lib/jasmine-
2.4.1/jasmine_favicon.png">
   <link rel="stylesheet" href="lib/jasmine-2.4.1/jasmine.css">
   <script src="lib/jasmine-2.4.1/jasmine.js"></script>
   <script src="lib/jasmine-2.4.1/jasmine-html.js"></script>
   <script src="lib/jasmine-2.4.1/boot.js"></script>
      <!-- include source files here... -->
   <script src="src/Player.js"></script>
   <script src="src/Song.js"></script>
   <!-- include spec files here... -->
   <script src="spec/SpecHelper.js"></script>
   <script src="spec/PlayerSpec.js"></script>
</head>
<body></body>
</html>
```

To concentrate on what Jasmine is capable of, I am created a simple JS file slider.js with function we will unit-test that function.

```
vari=0;
var imagefiles=['rbags4.jpg','rbags5.jpg','rbags7.jpg','rbags9.jpg','rbags12.jpg'];
window.setInterval(startSlider,1500);
```

```
functionstartSlider()
{
        document.getElementById("topimg").src="images/slider/"+imagefiles[i];
        i++;
        if(i>4)
        {
                i=0;
        }
}
```

And after adding file reference in SpecRunner.html, file content will be :

```
<!DOCTYPEhtml>
<html>
<head>
<metacharset="utf-8">
<title>Jasmine Spec Runner v3.1.0</title>

<linkrel="shortcut icon"type="image/png"href="lib/jasmine-3.1.0/jasmine_favicon.png">
<linkrel="stylesheet"href="lib/jasmine-3.1.0/jasmine.css">

<scriptsrc="lib/jasmine-3.1.0/jasmine.js"></script>
<scriptsrc="lib/jasmine-3.1.0/jasmine-html.js"></script>
<scriptsrc="lib/jasmine-3.1.0/boot.js"></script>

<!-- include source files here... -->
<scriptsrc="scripts/login.js"></script>
<scriptsrc="scripts/slider.js"></script>

<!-- include spec files here... -->
<scriptsrc="Tests/sliderTests.js"></script>
</head>
<body>
</body>
</html>
```

Let's start writing unit tests for slider.js to better understand suite and specs. We will write thesespecs inTests/ sliderTests.js.

```
/// <reference path="../scripts/slider.js" />
//This will be called before running each spec
beforeEach(function () {
```

```
varim = '<div id="topimg" ></div>';
document.body.insertAdjacentHTML('afterbegin', im);
});
//This will be called after running each spec
afterEach(function () {
document.body.removeChild(document.getElementById("topimg"));
});
//This is test suite
describe('A suite', function () {
//Spec for startSlider operation
it('contains spec with an expectation', function () {
startSlider();
    });
            });
```

HTML Assertion

HTML Assertion is used to verify that the response contains correct HTML syntax or not using JTidy (HTML Syntax Checker). It will fail the test in case of improper HTML syntax response.

This is log file to collect the errors and warning of improper HTML syntax response.

```
line 4 column 9 - Warning: <meta> lacks "content" attribute
line 144 column 16 - Warning: unknown attribute "property"
line 144 column 54 - Warning: unknown attribute "property"
line 145 column 9 - Warning: <script> lacks "type" attribute
line 155 column 1 - Warning: <link> isn't allowed in <body> elements
line 156 column 1 - Warning: <link> isn't allowed in <body> elements
line 157 column 1 - Warning: <link> isn't allowed in <body> elements
line 158 column 1 - Warning: <style> isn't allowed in <body> elements
line 205 column 15 - Warning: <style> isn't allowed in <body> elements
line 209 column 15 - Warning: <style> isn't allowed in <body> elements
```