TP1: TeaStore Quality Assurance

Group 02

Chun-An Bau
2165883
chun-an.bau@polymtl.ca

Youssef Amine BenDiha
2113648
youssef-amine.ben-diha@polymtl.ca

Jakub Profota
2165556
jakub.profota@polymtl.ca

CONTENTS

***Abstract*—This document presents a quality plan for one of the subsystems of TeaStore [1] focusing on three quality characteristics of ISO 25010 standard: Functional Suitability, Reliability, and Maintainability. The latest version of TeaStore is evaluated based on presented quality metrics and compared with previous versions.**

I. Introduction

TeaStore is an open-source distributed micro-service reference and test application featuring five distinct services and a registry. This basic emulation of a web store is to be used in benchmarks and tests. As it is a reference application, quality is critical. A tool used for benchmarks and tests must be reliable and functional. In this paper, we evaluate our quality assurance plan on the Authentication service of the TeaStore, focusing on three quality characteristics of ISO 25010: Functional Stability, Reliability, and Manageability.

In Functional Stability quality characteristics, we focused on the Completeness and Correctness of the Authentication service by evaluating the quality of the subsystem in runtime by interacting with the user interface, both manually and automatically. For the Reliability characteristics, we focused on the Maturity and Availability of the service by stressing the whole system with load generator benchmarks and evaluating response times and downtimes. Finally, in Maintainability quality characteristics, we focused on the Modularity and Testability, examining the codebase by introducing a modularity index and checking the test coverage of included unit tests. These metrics were evaluated on the latest version of TeaStore and then compared to older versions.

II. Quality Plan

A. Quality Goals

The specified quality characteristics and their respective sub-characteristics of the Authentication subsystem are presented in Table 1. For Completeness, the metric is the compliance with the specification. The subsystem must allow users to log in, log out, and see the personal data connected to their account, like name and email address. For Correctness, the user interface is tested through unit tests. No test can fail. For Reliability, a quantitative metric is used to evaluate the performance of the system under heavy load. Meantime of delay when accessing the subsystem cannot surpass 3 seconds. This bound is rather generous because the user usually only logins and logouts. For Availability, the Authentication subsystem under heavy load cannot be down for more than 1% of the stress test duration. For Modularity, we found a paper that introduces a Modularity Index obtained by analysis of the source code. We use this approach. Finally, for Testability, percentual code coverage of the included unit tests is obtained. We believe at least 70% of the subsystem codebase should be covered as this is a reference application.

B. Quality Assurance Strategies

TABLE II
Quality Assurance Strategies of the Authentication subsystem

Scope	Stage	Roles involved
Completeness	System testing	Clients
Correctness	Development	Developers
Maturity	System testing	Third party
Availability	System testing	Third party
Modularity	Development	Developers
Testability	Development	Developers

Both Functional Suitability and Reliability are strongly dependent on the runtime rather than the code. Therefore, most of the quality strategies we introduce in Table 2 are performed in the system testing stage. The product is not functionally suitable, nor can it be adequately benchmarked before if it is finished. The Completeness quality sub-characteristic is usually tested by clients, who also determine the specification of the product. In this case, clients are the developers. However, this strategy should be evaluated in the system testing stage nonetheless. The Correctness is usually tested throughout the development process. The developers should properly test every change to the subsystem. Once the subsystem is finished and not being changed, multiple running of unit tests is obsolete. Both the Maturity and Availability rely on system testing strategies. As this is a reference application, we can expect third parties to conduct the most performance-heavy testing. The developers should actively pursue Modularity throughout the development process. Finally, the Testability is closely related to Correctness and, as such, should be evaluated throughout the development process. Also, new unit tests are usually built in or even before the actual process of development.

III. Verification of the Quality Characteristics

A. Verification Methods

Following are the verification methods we used to evaluate the quality sub-characteristics. We focused mainly on publicly available and preferably open-source tools to aid us. All steps necessary to reproduce our results are included.

TABLE I
Quality Goals of the Authentication subsystem

Quality characteristics	Sub-characteristics	Quality measure	Objective
Functional Suitability	Completeness	Compliance with the specifications	All the specified functions of the system must be present
	Correctness	Unit Tests	All tests must pass
Reliability	Maturity	Delay mean time	Meantime 3 seconds maximum
	Availability	Downtime	Downtime 1% maximum under load
Maintainability	Modularity	Modularity index	Index reaches 0.5
	Testability	Code coverage	Test cases reach 70% of code coverage

TABLE III
QUALITY ASSURANCE STRATEGIES OF THE AUTHENTICATION SUBSYSTEM

Quality Sub-characteristics	**Tool/Method/Approach**
Completeness	Manual check
Correctness	Robot Framework
Maturity	LIMBO/JMeter
Availability	LIMBO/JMeter
Modularity	Modularity Index
Testability	JUnit

1) Completeness: We deployed the TeaStore in Docker the way it is documented in TeaStore's wiki page [2]. We first deployed in a single container to get familiar with the Docker command-line and then proceeded to run multiple single service containers. We used the 127.17.0.1 loopback IP address for all the services and successfully connected to the locally hosted website from a browser. We manually tested login, logout, user profile page, basket, and other website features.

2) Correctness: We first ran the included unit tests in root directory with MVN CLEAN TEST, then we decided to also test the runtime correctness with Robot Framework [3]. With this tool, we could automatically test the Authentication system through the user interface by directly writing our unit tests. The installation is straightforward. However, another library was needed to also test buttons like sign in and logout by emulating mouse clicks. For this functionality, Robot Framework uses the Selenium library [4]. Both are easily installed through pip [5].

3) Maturity: Our idea was to use one of the listed load generators to perform a stress test, but we were, unfortunately, unable to successfully deploy both of the tools. First, we tried to use JMeter [6]. However, both GUI and the command-line version exited with an error when we tried to run the profiles created by TeaStore authors. We used Ubuntu 21.04 64-bit GNU/Linux [7] operating system (OS) directly on hardware and in VMware Workstation 16 Pro [8] virtual machine manager, but to no avail. As our second option, we tried using the LIMBO HTTP [9] load generator. The deployment was successful, but we could not connect the director and the load generator to perform the benchmark. We tried deploying both of these on one OS and using the 127.17.0.1 loopback address. Then we deployed each system on its own virtual machine OS and tried the loopback address. Finally, we used two computers connected to the same private VPN created with ZeroTier [10]. Both machines saw each other, but the director and the load generator refused to connect in all the cases. For these reasons, we are unfortunately not able to list any performance-related results. The Reliability quality characteristic is not tested.

4) Availability: See III-A3.

5) Modularity: We used a formulation to measure the modularity level of software systems introduced in the paper by Andi Emanuel and Jazi Istiyanto [11]. The formulation was composed of Class Quality Value (C_Q), Package Quality Value (P_Q), and Software Architecture Value (S_A). We involved the first two metrics since we only analyzed the Authentication subsystem of the TeaStore application instead of the whole software architecture. According to the formulation, a higher value of P_Q indicates a higher level of Modularity. A package comprises several classes, and the Package Quality Value(P_Q) is calculated by the average class quality value C_Q. Three metrics determine the class quality value C_Q, where

$$C_Q = \frac{LOC_Q + F_Q}{2 \cdot LCOM4}$$

Definition of LOC_Q is

$$LOC_Q = 0.0125 \cdot NCLOC + 0.375 \ for \ NCLOC \leq 50$$

$$LOC_Q = (NCLOC - 50)^{-2.046} \ for \ NCLOC > 50$$

where NCLOC stands for Non Commenting Lines of Code. Definition of F_Q is

$$F_Q = 0.172 \cdot F + 0.171 \ for \ F \leq 5$$

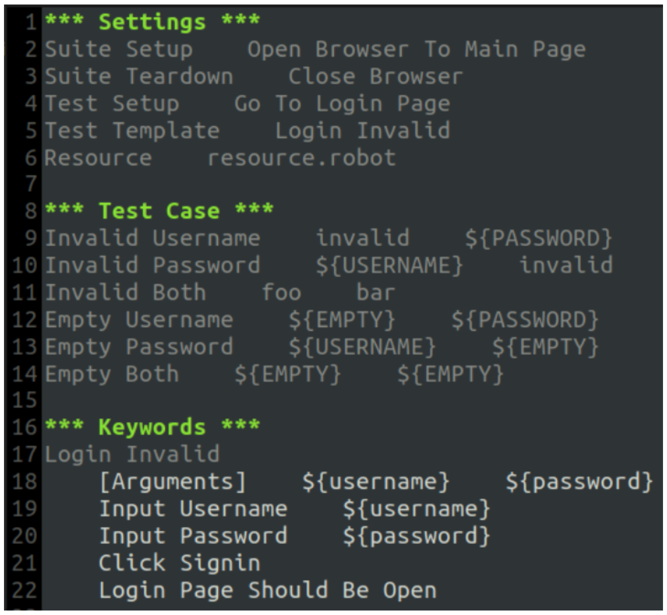
$$F_Q = (F - 4.83)^{-2.739} \ for \ F > 5$$

where F stands for Number of Function. Finally, a higher value of LCOM4 indicates lower maintainability. The value 1 is the ideal case which means that the class is highly cohesive.

6) Testability: In order to obtain the line coverage, we conducted JUnit [12] into our verification procedures. It is well-integrated with many common IDEs, such as IntelliJ, Android Studio, and Eclipse, which ease the verification. The coverage result will show up on the screen within minutes after launching the tests.

B. Verification Results

1) Completeness: The manual check of the Authentication subsystem functionality have found no issues. Everything works as expected.

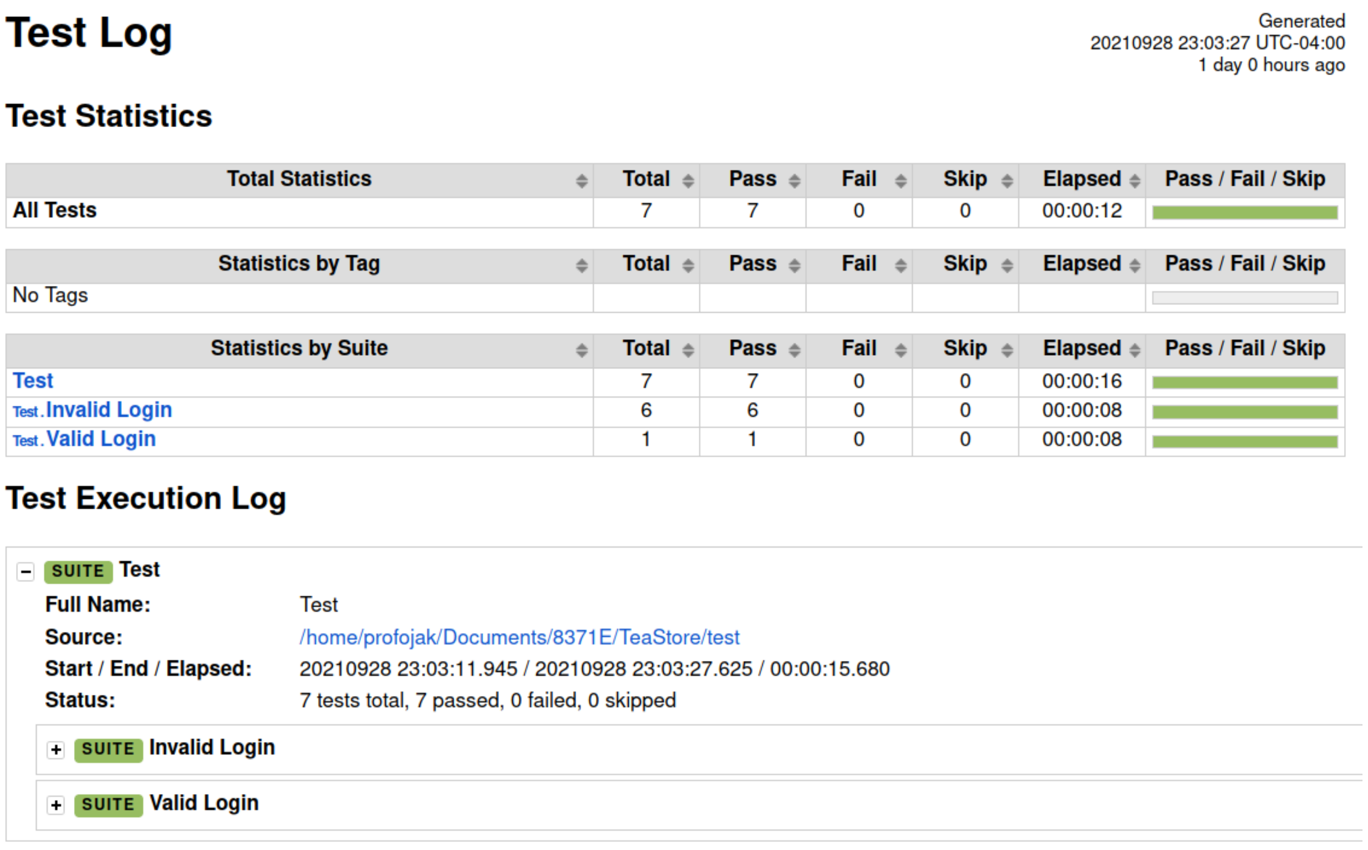


```
*** Settings ***
Suite Setup    Open Browser To Main Page
Suite Teardown    Close Browser
Test Setup    Go To Login Page
Test Template    Login Invalid
Resource    resource.robot

*** Test Case ***
Invalid Username    invalid    ${PASSWORD}
Invalid Password    ${USERNAME}    invalid
Invalid Both    foo    bar
Empty Username    ${EMPTY}    ${PASSWORD}
Empty Password    ${USERNAME}    ${EMPTY}
Empty Both    ${EMPTY}    ${EMPTY}

*** Keywords ***
Login Invalid
    [Arguments]    ${username}    ${password}
    Input Username    ${username}
    Input Password    ${password}
    Click Signin
    Login Page Should Be Open
```

Fig. 1. Invalid login Unit Test


Test Log

Generated
20210928 23:03:27 UTC-04:00
1 day 0 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	7	7	0	0	00:00:12	🟩

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Test	7	7	0	0	00:00:16	🟩
Test.Invalid Login	6	6	0	0	00:00:08	🟩
Test.Valid Login	1	1	0	0	00:00:08	🟩

Test Execution Log

SUITE Test

Full Name:	Test
Source:	/home/profojak/Documents/8371E/TeaStore/test
Start / End / Elapsed:	20210928 23:03:11.945 / 20210928 23:03:27.625 / 00:00:15.680
Status:	7 tests total, 7 passed, 0 failed, 0 skipped

SUITE Invalid Login

SUITE Valid Login

Fig. 2. Test Log of Robot Framework Unit Test

2) Correctness: Fig. 1 shows an example of one of the Robot Framework unit tests. Each unit test opens a browser and automatically goes through the user interface as specified. A delay between actions can be specified, but the TeaStore kept up with the delay of 0 milliseconds without any issues. A delay of 1500 ms was used for the first run to visually check the unit test's behavior, then 0 ms was set. Robot Framework outputted a test log file in HTML format seen in Fig. 2. All Authentication subsystem tests were successful.

3) Maturity: See III-A3.

4) Availability: See III-A3.

5) Modularity: There are eight classes under the Authentication subsystem. We measured the factors and calculated the quality value, as the Table 4 shows.

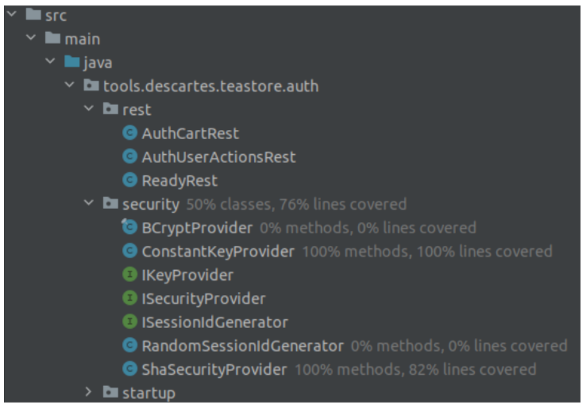

```
src
  main
    java
      tools.descartes.teastore.auth
        rest
          AuthCartRest
          AuthUserActionsRest
          ReadyRest
        security 50% classes, 76% lines covered
          BCryptProvider 0% methods, 0% lines covered
          ConstantKeyProvider 100% methods, 100% lines covered
          IKeyProvider
          ISecurityProvider
          ISessionIdGenerator
          RandomSessionIdGenerator 0% methods, 0% lines covered
          ShaSecurityProvider 100% methods, 82% lines covered
        startup
```

Fig. 3. Unit Test code coverage of the Authentication subsystem

TABLE IV
MODULARITY INDEX OF THE AUTHENTICATION SUBSYSTEM

Class	$NCLOC$	LOC_Q	F	F_Q	$LCOM4$	C_Q
AuthCartRest	40	0.875	3	0.687	1	0.781
AuthUserActionsRest	46	0.95	4	0.859	1	0.9045
ReadyRest	1	0.3875	1	0.343	1	0.3653
BCryptProvider	1	0.3875	2	0.515	1	0.4513
ConstantKeyProvider	1	0.3875	1	0.343	1	0.3653
RandomSessionIdGenerator	1	0.3875	1	0.343	1	0.3653
ShaSecurityProvider	38	0.85	5	1.031	1	0.9405
AuthStartup	5	0.4375	3	0.687	2	0.5623
Average (P_Q)						0.5568

6) Testability: There are three sub-directories, rest, security, and startup, under the directory of our target Authentication subsystem. The project provided some test cases for security, and the line coverage reached 76%. However, rest and startup coverage is 0% since there is no test case for the sub-directories, making the average line coverage only 25.3%.

IV. Version Comparison

TABLE V
QUALITY ASSURANCE STRATEGIES OF THE AUTHENTICATION SUBSYSTEM

Quality Sub-characteristic	**Results**	
	Version 1.0.0	**Version 1.4.0**
Completeness	Complete	Complete
Correctness	All tests passed	All tests passed
Maturity	See III-A3	See III-A3
Availability	See III-A3	See III-A3
Modularity	0.4056	0.5568
	Version 1.4.0	**Development branch**
Testability	rest: 0% security: 76% startup: 0%	rest: 73% security: 82% startup: 83%

1) Completeness: The old version is available at Docker Hub [13]. There is no difference between versions. The user interface looks identical and the Authorization subsystem works without issues.

2) Correctness: Same unit tests were run on the old version 1.0.0. All tests passed without any issues. The testing took the same time as in the newest version.

3) Maturity: See III-A3.

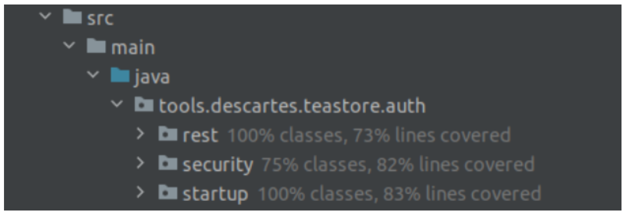
4) Availability: See III-A3.

5) Modularity: We conducted multiple Modularity metric evaluations on different commits and chose the most interesting one we found (commit hash 301CA2E). As shown in the Table 6, the Modularity level was apparently lower than the current version (see Table 4), which means that the Modularity level increased in the evolution of TeaStore.

6) Testability: We forked the development branch [14], rewrote some unit tests and evaluated the line coverage again. After that, the average line coverage of the Authentication subsystem is around 80%, which is significantly better than the previous approach (see Table 5).

V. Conclusion

This document presented the quality plan of the Authentication subsystem for three quality characteristics of the ISO 25010 standard, performed their evaluation, and compared the newest and old versions of the TeaStore reference application. According to the Functional Suitability testing, the TeaStore had all its features it has today implemented in the first version published. That means all the versions that followed only focused on the backend. Our Maintainability characteristic evaluation clearly shows that related aspects of the software were improved throughout the evolution. Modularity improved, the Testability significantly increased. We can also see the change in the implementation as both versions have a different number and names of classes tested in Modularity verification. It would be interesting to see if this change has any performance implications, but we were, unfortunately, unable to run benchmarks even though detailed instructions were presented on the wiki page.



```
src
  main
    java
      tools.descartes.teastore.auth
        rest 100% classes, 73% lines covered
        security 75% classes, 82% lines covered
        startup 100% classes, 83% lines covered
```

Fig. 4. Unit Test code coverage of the Authentication subsystem of the development branch

References

[1] https://github.com/DescartesResearch/TeaStore
[2] https://github.com/DescartesResearch/TeaStore/wiki/Getting-Started#run-teastore-containers-using-docker
[3] https://robotframework.org/
[4] https://robotframework.org/SeleniumLibrary/
[5] https://pypi.org/project/pip/
[6] https://jmeter.apache.org/download_jmeter.cgi
[7] https://ubuntu.com/
[8] https://www.vmware.com/products/workstation-pro.html
[9] https://github.com/joakimkistowski/HTTP-Load-Generator
[10] https://www.zerotier.com/
[11] Emanuel, Andi and Istiyanto, Jazi: Modularity Index Metrics for Java-Based Open Source Software Projects; International Journal of Advanced Computer Science and Applications, pp. 52-58, November 2011
[12] https://junit.org/junit5/
[13] https://hub.docker.com/r/descartesresearch/teastore-all/tags
[14] https://github.com/bauuuu1021/TeaStore

TABLE VI
MODULARITY INDEX OF THE AUTHENTICATION SUBSYSTEM IN THE OLD VERSION

Class	$NCLOC$	LOC_Q	F	F_Q	$LCOM4$	C_Q
AuthCartRest	39	0.8625	3	0.687	3	0.2583
AuthUserActionsRest	46	0.95	4	0.859	4	0.2261
ConstantKeyProvider	1	0.3875	1	0.343	1	0.3653
RandomSessionIdGenerator	1	0.3875	1	0.343	1	0.3653
ShaSecurityProvider	38	0.85	5	1.031	1	0.9405
AuthStartup	4	0.425	3	0.687	2	0.278
Average (P_Q)						0.4056