Test Report

Table of Contents

Test reports Legacy Requirement UC1....2
- Test case 1.1 Start server main scenario....2
- Test case 1.2.1 Start server modified main scenario input resource1....2
- Test case 1.2.2 Start server modified main scenario input “resource 4”....2
- Test case 1.2.3 Start server modified main scenario port boundary test....3
- Test case 1.3 Start alternative scenario 4a....4
- Test case 1.4 Start alternative scenario 4b....4
- Test case 1.5 Start alternative scenario 4c....4

Test reports Legacy Requirement UC2....5
- Test case 2.1.1 Stop server with input “ctrl+c”....5
- Test case 2.1.2 Stop server with input “Stop”....5
- Test case 2.1.3 Stop server with input “stop”....5
- Test case 2.1.4 Stop server with input “ctrl+z”....5

Test reports Legacy Requirement UC3....7
- Test case 3.1 Request shared resources main scenario....7
- Test case 3.2 Request shared resources alternate scenario 2a....7
- Test case 3.3 Request shared resources alternate scenario 2b....7
- Test case 3.4.1 Request shared resources alternate scenario 2c using request with invalid header....7
- Test case 3.4.2 Request shared resources alternate scenario 2c using request with invalid method....7
- Test case 3.5 Request shared resources alternate scenario 2d....8

Test reports Legacy Requirements Supplementary Specification....9
- Test case 4.1 Req1 Supplementary Specification from legacy requirements....9
- Test case 4.2 Req2 Supplementary Specification from legacy requirements....10
- Test case 4.3 Req4 Supplementary Specification from legacy requirements....10
- Test case 4.4 Req5 Supplementary Specification from legacy requirements....11

Test-cases concerning the maintainability of “My web server”....12
- Test-case 5.1 Test the maintainability of “My webserver”....12

Test-cases concerning the usability of “My web server”....13
- Test-case 6.1 Test the usability of “My webserver”....13

Test report compatibility between “My web server” and java versions....14
- Test-case 7.1....14

Test-cases concerning compatibility of “My web server” and common used OS on IoT devices....15
- Test-case 8.1 Windows10 IoT....15
- Test-case 8.2 Ubuntu Core....15
- Test-case 8.3 RIOT....15

Experimental test-cases....16
- Test-case 9.1....16

Traceability Matrix....17
- Table1, Legacy Requirements Use-cases....17
- Table2, Legacy Requirements Supplementary Specification....18
- Table3, “My Web Server” and SDC:s Goal 1....18

Test reports Legacy Requirement UC1

Test case 1.1 Start server main scenario

Result: Failed

Notes:
When exercising this test-case a error message displays in the terminal “This program should only have one or two arguments”. After exercising exploratory testing and code reviewing it become clear that the server can be started but that you have to input the port number and resource folder directly in the terminal when you run the terminal. This finding to start the web server is described in the following manual and will be used when exercising other test-cases that depends on that the web-server needed to be started.

Manual to start the web-server (Extracted from the new learning about the system)

To start a web server run in the terminal following command

$ java -jar MyWebServer.jar <port_number> <path_to_resource_folder>

Test case 1.2.1 Start server modified main scenario input resource1

Result: Succeed and failed

Notes
The server started at the input port number and the web-browser showed the index.html in the resource folder. But no file access log was written to. After further investigation of source code the program do not seem to posses the functionality to write to any log file but this I think can be implemented if necessary.

Test case 1.2.2 Start server modified main scenario input “resource 4”

Result: Failed

Notes
The web-server display in the terminal that only two arguments is valid to start the web-server. The web-server seem to interpret the space in the input argument “resource 4” as two argument. This bug should be solved to make the server more user friendly.

Test case 1.2.3 Start server modified main scenario port boundary test

Test result for 8 different port numbers

- Port 0 should NOT start: True the-server do NOT start
- Port 1 should start: False the-server do NOT start, the-serve claim that the port is taken
- Port 1023 should start: False the-server do NOT start, the-serve claim that the port is taken
- Port 1024 should start: True the server do start on the input port-number
- Port 49151 should start: True the server do start on the input port-number
- Port 49152 should start: True the server do start on the input port-number
- Port 65535 should start: True the server do start on the input port-number
- Port 65536 should NOT start: True the-server do NOT start

Note
That the server did not started on port that is “system or well-known ports” was not that the-server claimed to happen. This should be fixed in the source code of the program so that the server can start on ports that the server is claiming to be valid ports.

Test case 1.3 Start alternative scenario 4a

Result: Failed

Note:
Expected result: Socket 9000 was taken
Actual result: Port is taken

The result of this test case can be discussed, if output “Socket <port_number>” is preferred over output “Port is taken” I think this can be implemented in the source code.

Test case 1.4 Start alternative scenario 4b

Result: Failed

Notes:
Expected result: system display “No access to folder resource3”
Actual result: system display “HTTP server started” but browser display 404 Not found

The result of this test case can be discussed and further investigated if this is a security risk or if this result is an acceptable result. If found that this is not accessible I think the the expected result “No access to folder resource3” can be implemented in the source code.

Test case 1.5 Start alternative scenario 4c

Result: Failed

Notes:
Expected result: system display “Cannot write to server log file log.txt”

From previous test it have been showed that in the current state of the system the system do not print to any log files. If log file is NOT necessary this result is probably a result that we want in the system but if a log file is decided to be implemented I think the expected result in this test should be implemented so that the user get alerted when nothing is printed to the log file.

Test reports Legacy Requirement UC2

Test case 2.1.1 Stop server with input “ctrl+c”

Result: Failed

Notes:
Expected result: The system stops the web-server and presents that the webserver has been stopped
Actual result: The system stop the web-server but nothing was displayed in the terminal

The result of the the test can be test if it is enough if the web server stop the server and nothing is displayed to user or if the user also should be informed that the server have been stopped.

Test case 2.1.2 Stop server with input “Stop”

Result: Failed

Notes:
Expected result: The system stops the web-server and presents that the webserver has been stopped
Actual result: The web-server is not stopped

Test case 2.1.3 Stop server with input “stop”

Result: Succeeded

Notes:
Expected result: The system stops the web-server and presents that the webserver has been stopped
Actual result: The system stops the web-server and presents following in the terminal

- $ HTTP Server Accept thread stopped
- $ HTTP Server stopped

In all cases the user stop the server in the terminal I think this is a user friendly way to tell the user that the the server have been stopped. In all other way a user can stop a server in the terminal I think this message or something similarly could be displayed to the user.

Test case 2.1.4 Stop server with input “ctrl+z”

Result: Failed

Notes:
Expected result: The system stops the web-server and presents that the webserver has been stopped
Actual result: The system stop the web-server but nothing was displayed in the terminal

Alert:
After testing this test two times in a row the server could not be started the second time due the web-server claimed and displayed that the port is taken. I think further investigation of this is needed to be certain that the web-server stop the current port if the server stop in an unexpectedly way (like ctrl+z). If you would like to stop the open port an you use Ubuntu 20.04 or some similarly Linux distribution run the following command in the terminal

- $ fuser -k 9000/tcp

Test reports Legacy Requirement UC3

Test case 3.1 Request shared resources main scenario

Result: Succeeded

Test case 3.2 Request shared resources alternate scenario 2a

Result: Succeeded

Test case 3.3 Request shared resources alternate scenario 2b

Result: Failed

Notes:
Expected result: 403
Actual result: 404 not found

Notes: This test result can be discussed because if status code 403 need to be implemented or if the status code 404 is an acceptable status code for this scenario.

Test case 3.4.1 Request shared resources alternate scenario 2c using request with invalid header

Result: Succeeded

Test case 3.4.2 Request shared resources alternate scenario 2c using request with invalid method

Result: Succeeded

Test case 3.5 Request shared resources alternate scenario 2d

Result:

After code reviewing I could find package “response” this package contain classes

- HTTP200OKFileResponse
- HTTP400BadRequest
- HTTP403Forbidden
- HTTP404FileNootFoundResponse
- HTTP405MethodNotSupportedResponse

This classes name and content strongly indicates that the only response status code currently handled by the web-server is

- 200 OK
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Supported

Status code 500 Internal Server Error should be implemented in the source code to get higher usability.

Test reports Legacy Requirements Supplementary Specification

Test case 4.1 Req1 Supplementary Specification from legacy requirements

This test show number of errors. A request is decided to be an error if the request response time is over 100 ms. The value used in this load-tests is described in document testCases.pdf test-case 4.1. The LATEST result is also logged to the log files,

- log_load_easy.csv (show all requests),
- log_load_medium.csv (show all requests),
- log_load_hard.csv (show failed requests),
- log_load_extreme.csv (show failed requests).

Result load test easy show errors

- Attempt 1: 0 %
- Attempt 2: 0 %
- Attempt 3: 0 %
- Attempt 4: 0 %
- Attempt 5: 0 %

Result load test medium show errors

- Attempt 1: 25 %
- Attempt 2: 28 %
- Attempt 3: 30 %
- Attempt 4: 30 %
- Attempt 5: 18 %

Result load test hard show errors

- Attempt 1: 65 %
- Attempt 2: 51 %
- Attempt 3: 47 %
- Attempt 4: 28 %
- Attempt 5: 44 %

Result load test extreme show errors

- Attempt 1: 78 %
- Attempt 2: 93 %
- Attempt 3: 51 %
- Attempt 4: 31 %
- Attempt 5: 36 %

Test case 4.2 Req2 Supplementary Specification from legacy requirements

Result status code:
Expected result based on static code reviewing of source code

- GET expected 200, actual 200
- HEAD expected 501 Method Not Implemented, actual 405 Method Not Supported
- POST expected 501 Method Not Implemented, actual 405 Method Not Supported
- OPTIONS expected 501 Method Not Implemented, actual 405 Method Not Supported
- PUT expected 501 Method Not Implemented, actual 405 Method Not Supported
- DELETE expected 501 Method Not Implemented, actual 405 Method Not Supported
- TRACE expected 501 Method Not Implemented, actual 405 Method Not Supported
- CONNECT expected 501 Method Not Implemented, actual 405 Method Not Supported

The test show that the web-server only implement http method GET of the tested methods this was expected after the static code reviewing. The result from methods not implement by the web-server returned with status code “405 Method Not Supported”, this I think can be updated in the source code if status code “501 Method Not Implemented” is preferred. You could argue that the method HEAD must be implemented in the web-server if you strictly follow RFC 2616, this also I think can be implemented in the source code.

Test case 4.3 Req4 Supplementary Specification from legacy requirements

Result: Failed

Expected: GPL-2.0 (General Public License, version 2)
Actual: MIT (Massachusetts Institute of Technology license)

“The MIT License (MIT)

Copyright (c) 2014 Daniel Toll

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.”

Note
I would recommended that this should be closely examined by the SDC to see if the above metioned license is compatible with the goals SDC have with the the web-server “My web server”.

Test case 4.4 Req5 Supplementary Specification from legacy requirements

Result: Failed

Note
From previous test and code reviewing it have been find that in the current state of the web-sever the web-server does not write to any log file. And thereby this could not be tested. If this is an element that is desired in the web-server, I think this can be implemented.

Test-cases concerning the maintainability of “My web server”

Test-case 5.1 Test the maintainability of “My webserver”

After conducting code reviewing of the legacy source code, it can be established that the code is well-structured and well-tested, the code has a total of 57 unit tests, a small number of this are integration tests. The tests have a method coverage and statement coverage of almost 100%, the only method that is not tested is the main method in class HTTPServerConsole. Following are notices about some test methods that have been investigated more due to to failing or that they do not test anything (out-comment tests).

Failing tests

If the server is running and these automatic tests are exercised, two of these tests show that they are failing

- Failing test-method 1: SocketClientTest.testGetFromOnlineServer()
- Failing test-method 2: ConsoleViewTest.testPortTaken()

However “Failing test-method1” can be rewritten to test the current resource folder, port-number and IP-address that that are currently running (this test should be moved the integration tests) and the other of them “Failing test-method 2” can be fixed by using “System.lineseparator()” instead of “\ n”.

Out-comment tests

After further code reviewing of the tests to tests were found with out comment tests, theses two test was

- Out comment test-method 1: PortTest.testPortOk
- Out comment test-method 1: HTMLFileResponseTest.testWriteResonse

Out-comment method 1 could be tested with assert method “assertDoesNotThrow()” if wanted but then the testing library must be uppdated to JUnit 5, some other solution may also exist to solve this.

Out-comment method 2 could is partially out-comment, the method have one verify that is not out-comment and one that is out-comment. The method also have an assertion method that is out-comment in a for loop. The second verify method can be used in the test but when trying to test the assertion method some variable can not be compiled, more investigation is needed.

Conclusion

Further investigation of the code should be done to understand the code better, a recommendation is to put more time and resources to start experiment with the code and hence increase the knowledge of “My web browser”.

Test-cases concerning the usability of “My web server”

Test-case 6.1 Test the usability of “My webserver”

After performing more exploratory testing and from previous tests, it can be stated that the current state of "My web server" usability is low. The bugs that have been found in the previous tests should of course be fixed but also a more user-friendly functionality such as help functions and logs should be implemented in the system to increase the current usability. At the current state of “My web server” after you start server, the only thing you can do to interact with the server is to enter “stop” in the terminal and the server stop abruptly, I do not think this is enough to deploy this web server on IoT devices. After previous code review, it has been established that the program is in relatively good condition and an improvement of the current CLI should be possible to implement.

A discussion should also be made about whether a user-friendly CLI is sufficient to attract IoT developer or if the system also should offer some kind of GUI.

Test report compatibility between “My web server” and java versions

Test-case 7.1

Result

- My web-server and java version 7 : The web server could be started
- My web-server and java version 8 : The web server could be started
- My web-server and java version 9 : The web server could be started
- My web-server and java version 10 : The web server could be started
- My web-server and java version 11 : The web server could be started
- My web-server and java version 12 : The web server could be started
- My web-server and java version 13 : The web server could be started

This was a test that test if source code could be compiled with different java versions and if the server could be started with those java versions. For more thorough tests, previous tests in document testCases.pdf could also be performed for all different java versions.

Test-cases concerning compatibility of “My web server” and common used OS on IoT devices

Test-case 8.1 Windows10 IoT

Result:

My web browser is compatible with IoT

Test-case 8.2 Ubuntu Core

Result:

My web browser is compatible with Ubuntu Core

Test-case 8.3 RIOT

Result:

My web browser is compatible with RIOT

Experimental test-cases

Test-case 9.1

Result

The result from the automated test was as expected

Learning from experiment

More experiments and learning should be done to be able to use Postman and Newman as automation tools to be able to automate some of the previously performed tests. However, this first experiment indicates that these programs are powerful tools for automating previous tests. Automation of tests reduces time and errors that can occur when many tests are performed manually under repeated circumstances.

The Jenkins tool has also been explored and it seems to be, together with Postman and Newman, a great tool that can contribute to a more stable continuous integration to a project if that is desired.

My recommendation as a software tester is that we should aim to use these tools if further development of "My web server" is desired or in other projects in SDC, however, more time and resources are needed to learn these tools.

Traceability Matrix

Table1, Legacy Requirements Use-cases

This is a traceability matrix over the use-cases from the legacy requirements
M = Main Scenario in use cases
A[scenario] = Alternative scenario in use cases

Requirement	UC1 M	UC1 A[4a]	UC1 A[4b]	UC1 A[4c]	UC2 M	UC3 M	UC3 A[2a]	UC3 A[2b]	UC3 A[2c]	UC3 A[2d]
Test cases										
1.1	X									
1.2.1	X									
1.2.2	X									
1.2.3	X									
1.3		X								
1.4			X							
1.5				X						
2.1.1					X					
2.1.2					X					
2.1.3					X					
2.1.4					X					
3.1						X				
3.2							X			
3.3								X		
3.4.1									X	
3.4.2									X	
3.5										X

Table2, Legacy Requirements Supplementary Specification

This is a traceability matrix over the supplementary specification from the legacy requirements

Requirement	Req1	Req2	Req3*	Req4	Req5
Test cases					
4.1	X				
4.2		X			
4.3				X	
4.4					X

* It has previously been discussed in the document testPlan.pdf that Supplementary Specification Req 3 will not be tested. Instead the requirement have been modified to "My web server" should be compatible with common used operative systems on IoT devices. This requirement you can find in Table3 in column "Goal1 Obj4"

Table3, "My Web Server" and SDC:s Goal 1

This is a traceability matrix over the tests concerning SDC:s goal 1, *"To know if the found web-server "My web server" can be used on a wide range of IoT devices"*, this objective will be noted as "SDC Goal1" in this table. In document testPlan.pdf section "List off what should be tested" it have been described four sub objectives that is deduced from "SDC Goal 1" this objectives will be denotes as follows

- Obj1 = "My web server" should have high maintainability
- Obj2 = "My web server" should have high usability
- Obj3 = "My web server" should be compatible with different java versions.
- Obj4 = "My web server" should be compatible with common used operative systems on IoT devices

Requirement	SDC Goal1 Obj1	SDC Goal1 Obj2	SDC Goal1 Obj3	SDC Goal1 Obj4
Test cases				
5.1	X			
6.1		X		
7.1			X	
8.1				X
8.2				X
8.3				X