Test cases

Table of Contents

Setup

Common precondition test-cases

This is a common precondition for the test cases

- Testing environment Ubuntu 20.04 (could work with other OS)
- Working folder: test_folder
- Start a terminal at the path of folder “test_folder” this folder should contain

  - ✔ docker
  - ✔ JMeter
  - ✔ MyWebServer.jar (= executable archive file build from the legacy source code)
  - ✔ postman
  - ✔ resource1
  - ✔ resource2
  - ✔ resource3
  - ✔ “resource 4”
  - ✔ restrict_resource3.sh
  - ✔ RIOT
  - ✔ start_server_with_no_argument.sh
  - ✔ start_server_with_two_argument.sh
  - ✔ unrestrict_resource3.sh

JMeter precondition

- Install JMeter 5.3 for Ubuntu 20.04.
- Add bin folder in the installed JMeter to environment variable PATH
- Increase HEAP variable in file bin/jmeter.bat in installed JMeter to 2560m
- Start a terminal at the path of folder “test_folder/JMeter” this folder should contain

  - ✔ load_easy.jmx
  - ✔ load_hard.jmx
  - ✔ load_medium.jmx
  - ✔ load_easy.jmx
  - ✔ run_load_tests.sh

Postman precondition

- Install Postman for Ubuntu 20.04
- Install Newman for Ubuntu 20.04
- Start a terminal at the path of folder “test_folder/postman” this folder should contain
  - ✔ env_my_web_server.postman_environment.json
  - ✔ test_http_status_code.postman_collection.json
  - ✔ test_status_code.sh
-

Docker precondition

- Install docker for Ubuntu 20.04
- Start a terminal at the path of folder “test_folder/docker” this folder should contain
  - ✔ compile_and_run_my_web_server.sh
  - ✔ Dockerfile
  - ✔ DockerfileTemplate
  - ✔ resource1
  - ✔ start_docker_container_test_java_version.sh
- Download MyWebServer from Assignment 2 (Source from legacy code that is NOT compiled) and add that folder with exact name “MyWebServer” to the “test_folder/docker” folder. Folder “test_folder/docker” should know contain
  - ✔ compile_and_run_my_web_server.sh
  - ✔ Dockerfile
  - ✔ DockerfileTemplate
  - ✔ MyWebServer (Source from legacy code that is NOT compiled)
  - ✔ resource1
  - ✔ start_docker_container_test_java_version.sh

Test-cases Legacy Requirement UC1

Test case 1.1 Start server main scenario

Description: Test main scenario in UC1 (legacy requirements) to start the web server.

Precondition:

- ✔ Common precondition tests-cases

Post condition:

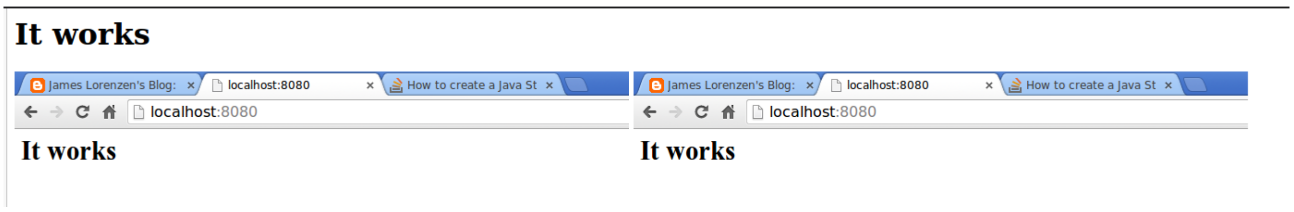
- A web server has been started
- A note in the access log was written, that the server was started

Test inputs

1. In the terminal run
   - ➢ $ ./start_server_with_no_argument.sh
2. In terminal input: 9000 resource1
3. In a web-browser enter localhost:9000

Expected output

1. In terminal system ask for port number and resource folder
2. The system start a web-server on the given port. In terminal the system presents that the web server have been started and the system also write a note in the access log.
3. In the web-browser



It works

It works

It works

Test case 1.2.1 Start server modified main scenario input resource1

Description: Start the web server following the manual to start the web-server described in the document testReport.pdf in the section “Test case 1.1 Start server main scenario”. This manual is the only currently known way to start the web-server.

Modified main scenario for starting the web-server

1. An administrator start the server in terminal by running the program with to arguments, argument one is the port number and argument two is the resource folder.
2. The system starts a web server on the given port and presents that the server was started and writes a note in the access log.

Precondition:

✔ Common precondition tests-cases

Post condition:

- A web server has been started
- A note in the access log was written, that the server was started

Test inputs

1. In the terminal run

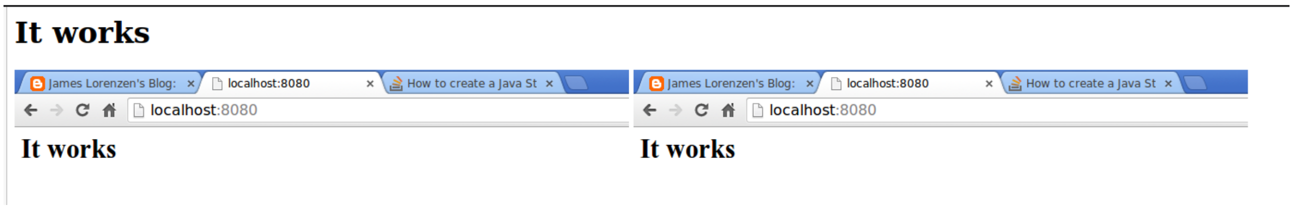
   ➢ $ ./start_server_with_two_argument.sh 9000 resource1

2. Run localhost:9000 in a web-browser

Expected output

1. The system start a web-server on the given port. In terminal the system presents that the web server have been started and the system also write a note in the access log.

2. The web-browser display following image


It works

It works

It works

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

Description: Start the web server using a resource folder with a space in the name.

Precondition:

- Common precondition tests-cases

Post condition:

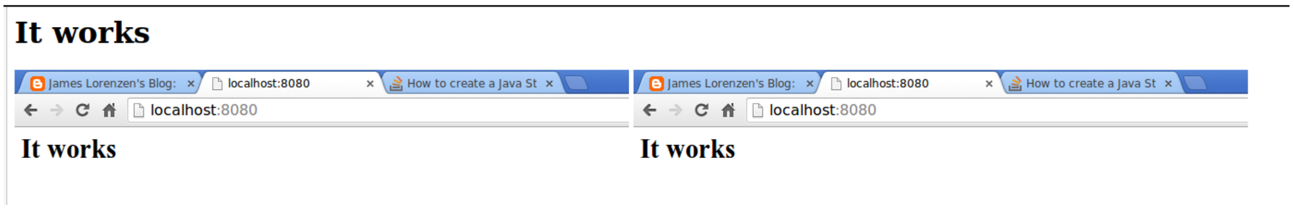
- A web server has been started

Test inputs

1. In the terminal run
   - ➢ $ ./start_server_with_two_argument.sh 9000 “resource 4“
2. Run localhost:9000 in a web-browser

Expected output

1. The system start a web-server on the given port. In terminal the system presents that the web server have been started and the system also write a note in the access log.
2. The web-browser display following image


It works

It works

It works

Test case 1.2.3 Start server modified main scenario port boundary test

Description: Start the web server with different port numbers. From previous testing the web-server displays that the valid port number is 1-65535. This test will test multiple boundary ports and see if the-server how the server handle those ports. This test-case can be seen as 8 separated test-cases but to minimize documentation waste this 8 test have compressed to one.

- Ports 0–1023 – system or well-known ports
- Ports 1024–49151 – user or registered ports
- Ports 49152–65535 – dynamic / private / ephemeral ports

Precondition:

- Common precondition tests-cases

Test inputs

OBS! The server must be stopped between every input

1. In the terminal run ./start_server_with_two_argument.sh 0 resource1
2. In the terminal run ./start_server_with_two_argument.sh 1 resource1
3. In the terminal run ./start_server_with_two_argument.sh 1023 resource1
4. In the terminal run ./start_server_with_two_argument.sh 1024 resource1
5. In the terminal run ./start_server_with_two_argument.sh 49151 resource1
6. In the terminal run ./start_server_with_two_argument.sh 49152 resource1
7. In the terminal run ./start_server_with_two_argument.sh 65535 resource1
8. In the terminal run ./start_server_with_two_argument.sh 65536 resource1

Expected output

1. Web-server do NOT start.
2. Web-server do start on the input port.
3. Web-server do start on the input port.
4. Web-server do start on the input port.
5. Web-server do start on the input port.
6. Web-server do start on the input port.
7. Web-server do start on the input port.
8. Web-server do NOT start.

Test case 1.3 Start server alternative scenario 4a

Description: Test alternative scenario 4a in UC1 to start the web server with a port number that the server is currently running on.

Precondition:

- Test case 1.2 “Start server modified main scenario”
- Open a second terminal with the precondition “common precondition tests-cases”

Postcondition:

- No new web server started

Test inputs

1. In the second terminal run
   - $ ./start_server_with_two_argument.sh 9000 resource2
2. Run localhost:9000 in a web-browser

Expected output

1. The system presents an error message: “Socket 9000 was taken”
2. Web browser show screen picture (resource1)

Test case 1.4 Start server alternative scenario 4b

Description: Test alternative scenario 4b in UC1 to start the web-server from a a restricted resource folder. Assume that restriction is equal with that the resource folder can not be read or written from.

Precondition:

- ✔ Common precondition tests-cases
- Folder resource3 should be restricted to not be able to read or write from. To set folder resource3 to this restriction run in the terminal
  - ➢ $ script restrict_resource3.sh

Postcondition:

- No new web server started

Test inputs

1. In the terminal run
   - ➢ $ start_server_with_two_argument.sh 9000 resource3

Expected output

1. The system presents an error message: “No access to folder resource3”

Note: run script unrestrict_resource3.sh to lift the restriction of folder resource3.

Test case 1.5 Start server alternative scenario 4c

Description: Test alternative scenario 4c in UC1 to start a web-server and the log.txt file could not be written to.

Precondition:

- ✔ Common precondition tests-cases

Test inputs

1. In the terminal run
   - ➢
     ```
     $ start_server_with_two_argument.sh 8080 resource1
     ```

Expected output

1. The system presents an error message: “Cannot write to server log file log.txt”

Test-cases Legacy Requirement UC2

Earlier testes have shown that there is no current log file and that the system do not write to any log file therefor the postcondition “A note in the access log was written, that the server was stopped” will be assumed to fail and NOT exercised in these tests.

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

Description: Test main scenario in requirement UC2 to stop the server after the server have been have started.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run enter ctrl+c and press enter
   OBS! The web-server must be running

Expected output

1. The system stops the web server and presents that the webserver has been stopped

Test case 2.1.2 Stop Server with input “Stop”

Description: Test main scenario in requirement UC2 to stop the server after the server have been have started.

Precondition:

- Test case 1.2 “Start server modified main scenario”
  OBS! The web-server must be running

Test inputs

1. In the terminal input “Stop” and press enter
   OBS! The web-server must be running

Expected output

1. The system stops the web server and presents that the webserver has been stopped

Test case 2.1.3 Stop Server with input “stop”

Description: Test main scenario in requirement UC2 to stop the server after the server have been have started.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal input “stop” and press enter
   OBS! The web-server must be running

Expected output

1. The system stops the web server and presents that the webserver has been stopped

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

Description: Test main scenario in requirement UC2 to stop the server after the server have been have started.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run “ctrl+z” and press enter
   OBS! The web-server must be running

Expected output

1. The system stops the web server and presents that the webserver has been stopped

Test-cases Legacy Requirement UC3

Modified requirement

Earlier testes have shown that there is no current log file and that the system do not write to any log file therefor the postcondition in UC3 “A note in the access log was written, that access happened with the request information and the result of the request.” will be assumed to fail and NOT exercised in these tests. Instead the focus of these testes is to see if the system produce expected status codes to the user in different scenarios.

- The web-server handle the request with succeed gives status code 200 OK
- The web-server can not handle the request because the webserver did not understand the request gives status code 400 Bad request
- The web-server can handle the request but refusing to fulfills it gives status code 403 Forbidden
- The web-server can not handle the request because the webserver can not find the shared resource folder gives status code 404 Not Found

Test case 3.1 Request shared resources main scenario

Description: Test main scenario in requirement UC3 that the system produce status code 200 OK if the the request is OK.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

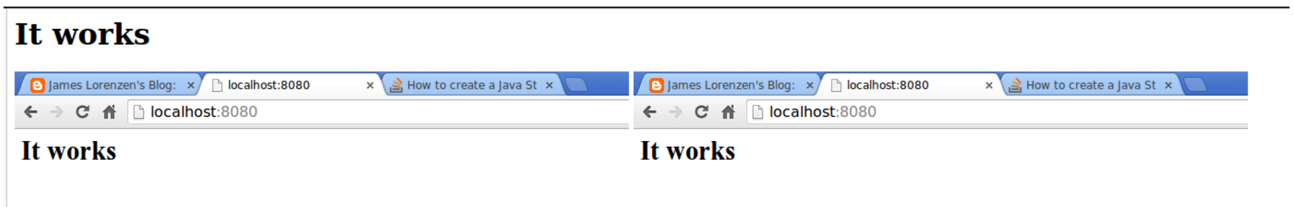
1. In the terminal run. OBS! The web-server must be running

   ➢ $ curl -i http://localhost:9000 | sed 1q

2. In a web-browser enter http://localhost:9000

Expected output

1. The terminal display HTTP/1.1 200 OK
2. The web-browser display



It works

It works

It works

Test case 3.2 Request shared resources alternate scenario 2a

Description: Test alternate scenario 2a in requirement UC3 that the system produce status code 404 Not Found if the the request resource can not be found.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run. OBS! The web-server must be running

   ➢
   ```
   $ curl -i http://localhost:9000/i_do_not_exist.html | sed 1q
   ```

Expected output

1. The terminal display HTTP/1.1 404 Not Found

Test case 3.3 Request shared resources alternate scenario 2b

Description: Test alternate scenario 2b in requirement UC3 that the system produce status code 403 if the resource request is outside the resource folder.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run. OBS! The web-server must be running

   ➢
   ```
   $ curl -i http://localhost:9000/../resource2/secret.html | sed 1q
   ```

Expected output

1. The terminal display HTTP/1.1 403

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

Description: Test alternate scenario 2c in requirement UC3 that the system produce status code 400 bad request if the the system can NOT understand the request because request is invalid or malformed.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run. OBS! The web-server must be running
   - $ curl -iH "Accept: text/html: text/html" http://localhost:9000/ | sed 1q

Expected output

1. The terminal display HTTP/1.1 400 bad request

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

Description: Test alternate scenario 2c in requirement UC3 that the system produce status code 400 bad request if the the system can NOT understand the request because request is invalid or malformed.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Test inputs

1. In the terminal run. OBS! The web-server must be running
   - $ curl -iX get http://localhost:9000/ | sed 1q

Expected output

1. The terminal display HTTP/1.1 400 bad request

Static review 3.5 Request shared resources alternate scenario 2d

Description: Test alternate scenario 2d in requirement UC3 that the system should produce that it has internal error , status code 500 Internal Server Error, if an internal error happened during a request. To perform this test that I will do code reviewing over the source code concerning status code management of the web-server.

Expected output

The expected result of the code reviewing is to find hat the web-server respond with status code 500 if the server has encountered a situation it doesn't know how to handle.

Test-cases Legacy Requirement Supplementary Specification

Test case 4.1 Req1 Supplementary Specification from legacy requirements

Description: Test that the web-server is responsive under high load. The software tester is using JMeter as tool to exercise load-testing on the web-server.

Precondition

- Common precondition tests-cases.
- In test_folder run ./start_server_with_two_argument.sh 9000 resource1.
- Close unnecessary programs, this to get better result from load-testing.
- JMeter precondition, see section “Setup”.

Parameters used in the tests

- users = number of user request
- ramp_up = How long to delay before next user request for example users = 100 and ramp_up = 1 gives the delay 0.01 second between users.
- duration_limit = the limit of how long a request should take, if this limit is exceeded the test will be assumed failed even if the request got handled. Google recommend a web-server response time to under 0.5 seconds, reference https://phoenixnap.com/kb/reduce-server-response-time, but because this tests is tested with the web-server on the same computer this test will use the response time limit set to 100 ms.
- loops = how many loops of the users

Parameters values in easy load test

- users = 10
- ramp_up = 1 (s)
- duration_limit = 100 (ms)
- loops = 1

Parameters values in medium load test

- users = 100
- ramp_up = 1 (s)
- duration_limit = 100 (ms)
- loops = 1

Parameters values in easy hard load test

- users = 500
- ramp_up = 1 (s)
- duration_limit = 100 (ms)
- loops = 1

<u>Parameters values in extreme load test</u>

- users = 1000
- ramp_up = 1 (s)
- duration_limit = 100 (ms)
- loops = 1

<u>Exercise test-case</u>

Run script run_load_tests.sh in 5 times and save value from terminal

Test case 4.2 Req2 Supplementary Specification from legacy requirements

Description: Test that the web-server follow the minimum requirements for HTTP 1.1 in this test case that the web-server can handle the methods as described in RFC 2616. The method GET and HEAD MUST be implemented and the other method is optional but the server SHOULD* return status code 405 (Method not allowed) if the methods is known by the server but not allowed for the requested resource, and 501 (Not Implemented) if the method is unrecognized or not implemented by the server, reference RFC 2616 page 36, https://www.ietf.org/rfc/rfc2616.txt

Methods to be tested based on RFC 2616

- GET
- HEAD
- POST
- OPTIONS
- PUT
- DELETE
- TRACE
- CONNECT

* “ SHOULD

This word, or the adjective "RECOMMENDED", mean that there
may exist valid reasons in particular circumstances to ignore a
particular item, but the full implications must be understood and
carefully weighed before choosing a different course.”- Reference RFC 2119, page 1,

https://tools.ietf.org/html/rfc2119

continues on next page...

Security check

After done a security check over the implemented functionality concerning this test using static code reviewing of the source code method “getResponse” could be find in class ResponseFactory.java.

```
public HTTPResponse getResponse(HTTPRequest request) throws IOException {
    HTTPRequest.Method method = request.getMethod();
    if (method == HTTPRequest.Method.GET) {
        try {
            File file = folder.getURL(request.getURL());
            return new HTTP200OKFileResponse(file, watcher, clientThread);
        } catch (FileNotFoundException e) {
            return new HTTP404FileNotFoundResponse(request.getURL());
        } catch (SecurityException e) {
            return new HTTP403Forbidden();
        }
    }

    return new HTTP405MethodNotSupportedResponse(request.getMethod());
}
```

this method indicates strongly to that the only http method that is currently implemented in the web server “My web server” is the http method GET.

Precondition:

- Test case 1.2 “Start server modified main scenario”

Input

In the terminal input. OBS! The-server must be running on localhost:9000

1. $ curl -i GET http://localhost:9000/ | sed 1q
2. $ curl -I HEAD http://localhost:9000/ | sed 1q
3. $ curl -i POST http://localhost:9000/ | sed 1q
4. $ curl -i OPTIONS http://localhost:9000/ | sed 1q
5. $ curl -i PUT http://localhost:9000/ | sed 1q
6. $ curl -i DELETE http://localhost:9000/ | sed 1q
7. $ curl -i TRACE http://localhost:9000/ | sed 1q
8. $ curl -i CONNECT http://localhost:9000/ | sed 1q

Expected output based on the result from the security check and RFC 2616

In terminal expected status code

1. 200 OK
2. 501 Method Not Implemented
3. 501 Method Not Implemented
4. 501 Method Not Implemented
5. 501 Method Not Implemented
6. 501 Method Not Implemented
7. 501 Method Not Implemented
8. 501 Method Not Implemented

Test case 4.3 Req 4 Supplementary Specification from legacy requirements

Description: Test that the source code is released under GPL-2.0

Description: Test that the source code is released under GPL-2.0 (General Public License, version 2).

Expected license

GPL-2.0 license

“*one line to give the program's name and an idea of what it does.*
Copyright (C) *yyyy name of author*

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.”

reference https://www.gnu.org/licenses/old-licenses/gpl-2.0-standalone.html

Test case 4.4 Req 5 Supplementary Specification from legacy requirements

Description: Test If the access log is viewable from a text editor

Approach:
Exploratory testing and documentation, the documentation can be find in the document testReport.pdf

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

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

Description:
Test how the maintainability of “My web server”. If further development of "My web browser" is to be done, it would be a great advantage if the legacy source code is well structured and that bugs can be easily fixed and that new functionality can be integrated if desired. This test aims to make this assessment of "My web server".

Approach:
Code reviewing and documentation, the documentation can be find in the document testReport.pdf. During the code review the main focus will be on how well the code is structured and how the code is unit-tested. If unit-test is in the source code these tests will be exercised and examined.

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

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

Description:
This test is an examination of how user-friendly the current state of "My web server is". This test is performed in this iteration of testing “My web server” to see if the usability is high enough to deploy this server on IoT devices.

Approach:
Exploratory testing and documentation, the documentation can be find in the document testReport.pdf

Test-cases concerning compatibility of “My web server” and different java versions

Test-case 7.1

<u>Description</u>

Test “My web server” with java version 7-13. These tests test if the source code could be compiled and if the web-server could be started with different java versions.

<u>Precondition</u>

- Common precondition tests-cases
- Docker precondition

<u>Input</u>

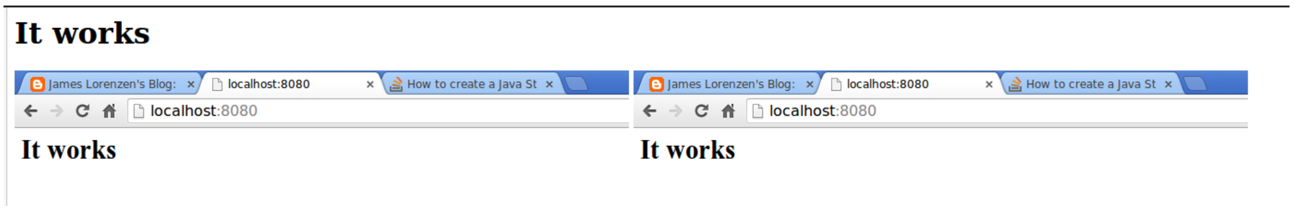
In the terminal path test_folder/docker run

1. $ ./start_docker_container_test_java_version.sh 7
   - Open a web-browser at localhost:9000
2. $ ./start_docker_container_test_java_version.sh 8
   - Open a web-browser at localhost:9000
3. $ ./start_docker_container_test_java_version.sh 9
   - Open a web-browser at localhost:9000
4. $ ./start_docker_container_test_java_version.sh 10
   - Open a web-browser at localhost:9000
5. $ ./start_docker_container_test_java_version.sh 11
   - Open a web-browser at localhost:9000
6. $ ./start_docker_container_test_java_version.sh 12
   - Open a web-browser at localhost:9000
7. $ ./start_docker_container_test_java_version.sh 13
   - Open a web-browser at localhost:9000

<u>OBS!</u> To stop a container you can run “./start_docker_container_test_java_version.sh STOP”, this will stop the currently created docker container.

<u>Expected result</u>

If the “My web server” is compatible with the tested java version the expected result is following result in a web-browser at localhost:9000



It works

It works

It works

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

Test-case 8.1 Windows10 IoT

Description

Test “My web server” with windows10 IoT. This is a test of “My web browser” to check if the web-server is compatible with Wndos10 IoT.

Precondition

- Set up a operative system windows10 IoT on VirtualBox
- Install Java11 on the operative system to be tested
- Download test_folder to the operative system to be tested

Test

- Perform the tests described in this document on the operating system windows10 IoT

Expected Result

Os is compatible

Test-case 8.2 Ubuntu Core

Description

Test “My web server” with Ubuntu Core. These tests test if the source code could be compiled and run on operative system Ubuntu Core.

Precondition

- Set up a operative system UbuntuCore on KVM follow instruction at https://ubuntu.com/download/kvm
- Install java on the operative system to be tested following instruction at https://github.com/jgneff/openjdk
- copy folder “test_folder/path” to the operative system to be tested
  - ➢ $ scp -P 8022 -r <path to folder “test_folder/path”>
- Log in to the operative system to be tested using ssh and also map port 9000
  - ➢ $ ssh -p 8022 -L 9000:localhost:9000 <username>@localhost
- Change to directory test_folder/docker in the operative system to be tested and run
  - ➢ $ sudo ./start_docker_container_test_java_version.sh 11

  This will start a container build with operative system Ubuntu Core and in this container “My web browser” starts on port 9000

Test
Perform the tests described in this document on the operating system Ubuntu Core

Expected Result

OS is compatible

Test-case 8.3 RIOT

Description

Test “My web server” with RIOT. This test test if the source code could be compiled and run on operative system RIOT.

Precondition

- Common precondition tests-cases
- Docker precondition
- Change to directory test_folder/RIOT this folder should contain compile_and_run_my_web_server.sh
    - Dockerfile
    - resource1
    - start_riot_container.sh
- Download MyWebServer from Assignment 2 (Source from legacy code that is NOT compiled) and add that folder with exact name “MyWebServer” to the “test_folder/RIOT” folder. Folder “test_folder/RIOT” should know contain
    - Dockerfile
    - MyWebServer (Source from legacy code that is NOT compiled)
    - resource1
    - start_riot_container.sh
- Run script in terminal
    - $ ./start_riot_container.sh

  This will start a container build with operative system RIOT and in this container “My web browser” start on port 9000

Test

Perform the tests described in this document on the operating system Ubuntu Core

Expected Result

OS is compatible

Experimental test-cases

Test-case 9.1

Description

This test case is an experiment on how previous test cases could be automated with Postman and Newman. The test-cases that has experiment with is test concerning http methods and the returning status code from “My web server”.

Precondition

- Common precondition tests-cases
- Postman precondition see section “Setup”
- In test_folder run ./start_server_with_two_argument.sh 9000 resource1.

Input

In terminal path t”est_folder/postman” run script

➢ $ ./test_status_code.sh

Expected result

These expected result is based on previous tests regarding “My web server” and http status code

- Test status code for http method GET : Should return 200
- Test status code for http method POST : Should return 405
- Test status code for http method DELETE : Should return 405
- Test status code for http method PUT : Should return 405
- Test status code for http method OPTIONS : Should return 405
- Test status code for http method GET with invalid path to resource folder : Should return 404
- Test status code for http method GET with path to resource folder outside root folder : Should return 403
- Test status code for http method POST with invalid body : Should return 400