Test Plan for Ataccama TestCalc application

Contents

Overview 2
  Test Environment for the UI tests 2
  Choosing test parameters 2
  Pairwise Independent Combinatorial Testing 3
  General Test Description 4
  Manual test cases examples 5
Test Automation Approach 6
  Technologies stack 6
  Test automation project overview 6
    UI and API testing 6
    Performance testing 7
  Project settings 8
    Setup 8
    Run UI and API tests 8
    Run JMeter tests 8
  Test reporting 8
Conclusion 9

Overview

Test Environment for the UI tests

According to https://gs.statcounter.com/, the most popular browser is Chrome (~70%). The following popular browser is Firefox (~10%). The UI tests will be run against Chrome and Firefox, and it will cover ~80% of all users.

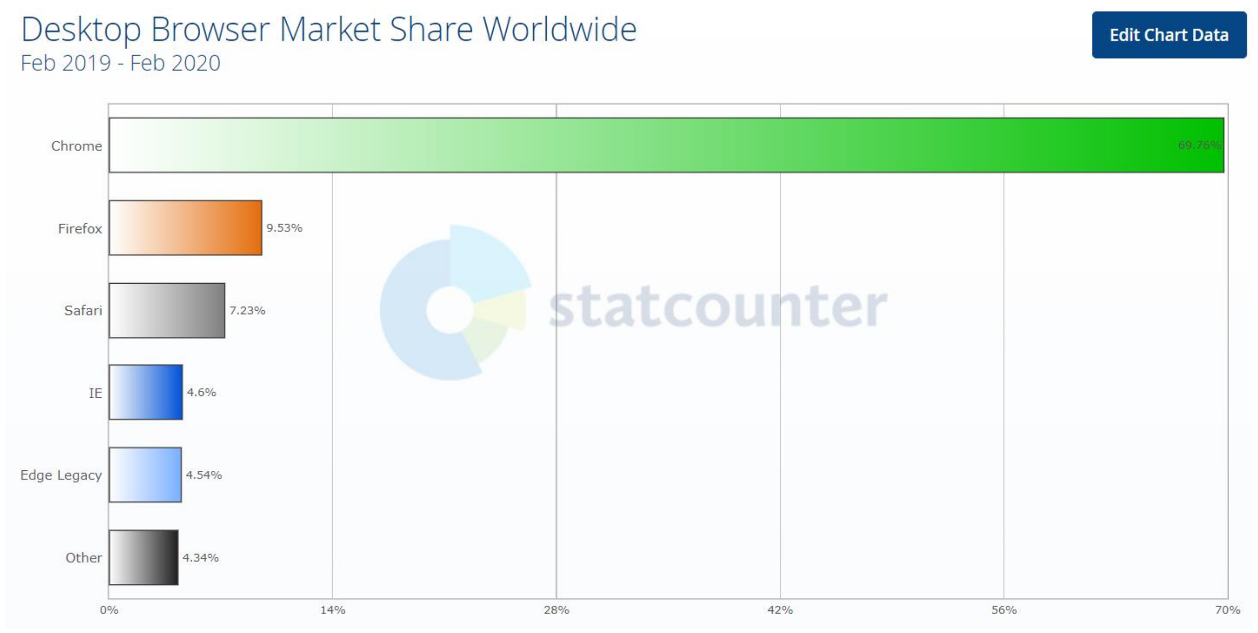


Fig. 1.1. Desktop browser usage statistics

Choosing test parameters

Value1: integer values from -2147483648 to 2147483647

Value2: integer values from -2147483648 to 2147483647

Operations: addition, subtraction, multiplication, division

The main challenge is to choose a valid set of test values to cover all risky combinations. To best way to achieve this is to use such Test Design techniques as Equivalence Partitioning and Boundary Value Analysis.

All data can be divided into the following equivalence classes:

1. **Below zero**. Values from -2147483648 to 1.
2. **Zero**. Contains only one value – 0.
3. **Above zero**. Values from 1 to 2147483647.
4. Integers with **leading zero**, e.g. 0000012345689 – zeroes should be removed automatically.

In addition, there can be considered a negative class, which contain values, causing the exception in the calculator.

5. **Negative**.
   - -2147483649, 2147483648 – values out of integer range
   - empty values

- non-integer values (e.g. 1.0)
- strings (e.g. “aBd%##@(%)@”)
- valid integers with trailing empty strings, e.g. “1111111 “
- zero division

Final classes set:

1. **Below zero**:

-2.147.483.648	boundary value
-46.340	$\sqrt{-2.147.483.648}$
-215	$\sqrt{46340}$
-15	$\sqrt{215}$
-1	boundary value

2. **Above zero**:

2.147.483.647	boundary value
46.340	$\sqrt{-2.147.483.647}$
215	$\sqrt{46340}$
15	$\sqrt{215}$
1	boundary value

3. **Zero:** 0
4. **Negative**:

-2147483649	Integer.MIN_VALUE - 1
2147483648	Integer.MAX_VALUE + 1
99.0	Non-integer
!@#$%^&*() as8o5	String
123 34 3	Invalid number with spaces
“ “	Empty string

Pairwise Independent Combinatorial Testing

For decreasing the number of test combinations was used **Pairwise Testing** technique.

Pairwise testing (a.k.a. all-pairs) is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most **two** factors.

Pairwise-generated test suites cover **all combinations** of two therefore are much smaller than exhaustive ones yet still very effective in finding defects.

For generating test parameters combinations was used tool PICT (Pairwise Independent Combinatorial Testing tool) by Microsoft. See models txt files in attachments.

PICT model for positive combinations:

Operations: addition, subtraction, multiplication, division

Value1: -2147483648, -46340, 215, 15, 1, 0, 2147483647, 46340, 215, 1

Value2: -2147483648, -46340, 215, 15, 1, 0, 2147483647, 46340, 215, 1

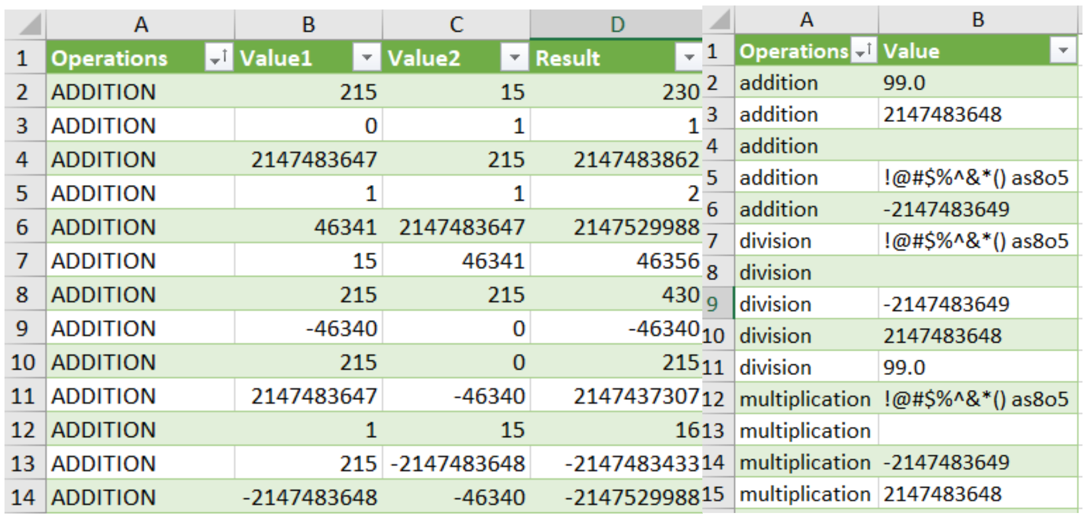
Generated combinations: 180

Generated tests: 100

PICT model for negative combinations:

Operations: addition, subtraction, multiplication, division

Value: -2147483649, 2147483648, 99.0, !@#$%^&*() as8o5, " "



	A	B	C	D
1	Operations	Value1	Value2	Result
2	ADDITION	215	15	230
3	ADDITION	0	1	1
4	ADDITION	2147483647	215	2147483862
5	ADDITION	1	1	2
6	ADDITION	46341	2147483647	2147529988
7	ADDITION	15	46341	46356
8	ADDITION	215	215	430
9	ADDITION	-46340	0	-46340
10	ADDITION	215	0	215
11	ADDITION	2147483647	-46340	2147437307
12	ADDITION	1	15	16
13	ADDITION	215	-2147483648	-2147483433
14	ADDITION	-2147483648	-46340	-2147529988

	A	B
1	Operations	Value
2	addition	99.0
3	addition	2147483648
4	addition	
5	addition	!@#$%^&*() as8o5
6	addition	-2147483649
7	division	!@#$%^&*() as8o5
8	division	
9	division	-2147483649
10	division	2147483648
11	division	99.0
12	multiplication	!@#$%^&*() as8o5
13	multiplication	
14	multiplication	-2147483649
15	multiplication	2147483648

Fig 1.3. Test data example

In addition, for automated testing were used random values.

NOTE: Test data spreadsheet is stored in *src/tests/resources* folder.

General Test Description

- Performing math operations (addition, subtraction, division, multiplication).
- Verify absence of errors and exceptions in case of valid input values.
- Verify stability of the application.
- Record operation result. Compare with the result from the external calculator (e.g. Windows Calculator).

Manual test cases examples

Preconditions for each test case:

- Installed Apache Tomcat web server
- Application .war file stored into Tomcat\webapps folder
- Web server is launched and works correctly
- Installed the Chrome browser

TC 1 - positive

Steps:

1. Open the browser, go to URL http://localhost:8080/testCalc/webUI
2. Select **addition** radio button
3. Value1 → Enter a valid integer number
4. Value2 → Enter a valid integer number
5. Click **Calculate**

Expected Results:

1. Application launched successfully
2. Operation radio button selected
3. **Value1** populated successfully
4. **Value2** populated successfully
5. Calculated result is correct, there is no exceptions

Repeat the test for all generated positive test conditions

TC 2 - negative

Steps:

1. Open the browser, go to URL http://localhost:8080/testCalc/webUI
2. Select **multiplication** radio button
3. **Value1** → Enter invalid value (empty string, non-integer, too big value, etc.)
4. **Value2** → Enter a valid integer number
5. Click **Calculate**

Expected Results:

1. Application launched successfully
2. Operation radio button selected
3. **Value1** populated successfully
4. **Value2** populated successfully
5. There is the exception thrown. Verify the exception has a valid message.

Repeat the test for all generated negative test conditions

Test Automation Approach

Technologies stack

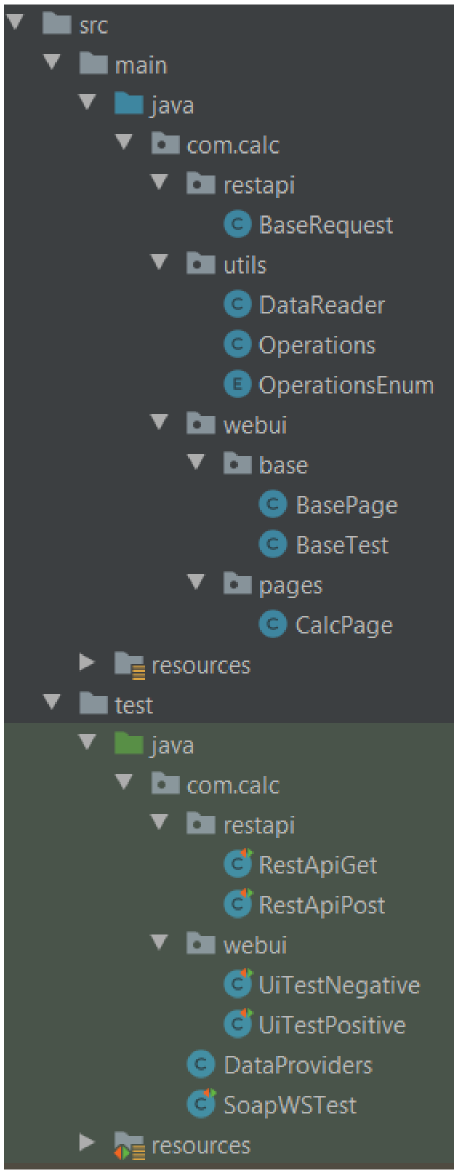
For the automation of the app was chosen **Java** as primary language. The main reason of this choice – the application under test also uses Java, and it has a lot of benefits, such as:

1. Same programming language fosters collaboration between developers and QA engineers.
2. Same programming language makes software environment setup easier: the same IDE, the same CI server, the same development environments, etc.

For the UI testing was used **Selenium WebDriver** https://www.selenium.dev/.

For the API testing was used **REST Assured** http://rest-assured.io/. This is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.

For performance and load testing was used **Apache JMeter** https://jmeter.apache.org/. The Apache JMeter™ application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.



```
src
  main
    java
      com.calc
        restapi
          BaseRequest
        utils
          DataReader
          Operations
          OperationsEnum
        webui
          base
            BasePage
            BaseTest
          pages
            CalcPage
    resources
  test
    java
      com.calc
        restapi
          RestApiGet
          RestApiPost
        webui
          UiTestNegative
          UiTestPositive
        DataProviders
        SoapWSTest
    resources
```

Test automation project overview

UI and API testing

main.java.com.calc

restapi – base class for Rest Api tests

utils – contains classes for reading test data from the Excel. Also contains enum and class for auxiliary operations.

webui – contains logic for UI tests

base – base classes for tests and pages

pages – page objects for the application.

Actually, the **testCalc** app has the only page, but this structure was created for easy extendibility.

resources – contains webdrivers executables.

<u>*test.java.com.calc*</u>

This package contains test classes for UI and Rest API. The SoapWSTest class is not fully implemented – unfortunately, I didn't manage to send any SOAP request due to the lack of experience in SOAP API testing.

restapi - contains API tests using REST-assured library

RestApiGet

Contains parametrized tests for GET requests, also has tests with random values. Covered both positive and negative conditions.

RestApiPostt

Contains parametrized tests for POST requests, also has tests with random values. Covered both positive and negative conditions.

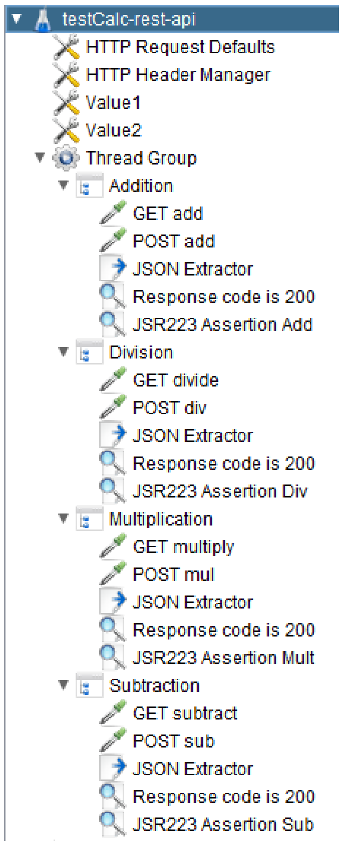
webui – contains tests for UI using Selenium Webdriver.

UITestNegative – parametrized tests for negative conditions.

UITestPositive – – parametrized tests for positive conditions.

DataProviders – class with static methods – TestNG data providers.

resources – contains Excel test data with PICT-generated test pairs.



- testCalc-rest-api
  - HTTP Request Defaults
  - HTTP Header Manager
  - Value1
  - Value2
  - Thread Group
    - Addition
      - GET add
      - POST add
      - JSON Extractor
      - Response code is 200
      - JSR223 Assertion Add
    - Division
      - GET divide
      - POST div
      - JSON Extractor
      - Response code is 200
      - JSR223 Assertion Div
    - Multiplication
      - GET multiply
      - POST mul
      - JSON Extractor
      - Response code is 200
      - JSR223 Assertion Mult
    - Subtraction
      - GET subtract
      - POST sub
      - JSON Extractor
      - Response code is 200
      - JSR223 Assertion Sub

Performance testing

For the performance testing used Apache JMeter tool. JMeter script *testCalc-rest-api.jmx* can be found in *test-jmeter* folder.

All requests use **random**-generated values for **Value1** and **Value2**. In case of performance testing using Excel data sheets is exhaustive.

• *HTTP Request Defaults* – has default URL for all requests. In our case it's 'localhost'
• *HTTP Header Manager* – default header for each request.
• *Value1* and *Value2* – random variables. Each value is regenerated for each new thread (user).
• *Thread Group* is a set of threads executing the same scenario. It is the base element for every JMeter test plan.
• *Simple Logic Controllers* – for each operation.

Each operation has GET and POST request, and additional elements:

- *JSON Extractor* to fetch the result of values calculation.
- *Response code assertion* – checks the response is OK
- *JSR223 Assertion* – checks the result of calculation.

To run tests from cm, launch the *run.cmd* file in the script root folder.

Project settings

Setup

1. Installed Java, added JAVA_HOME to environment variables.
2. Installed Maven, added MAVEN_HOME environment variables.
3. Add M2 variable to path.
4. Installed the latest versions of Chrome and Firefox browsers.
5. Download https://jmeter.apache.org/download_jmeter.cgi and install JMeter.
6. Add <Jmeter folder\bin> to the PATH.
7. Download the project source code.

Run UI and API tests

There are two three available ways to run tests:

1. Separately from the IntelliJ Idea.
2. Run TestNG configuration files from IntelliJ Idea:

a. *crossbrowser-testng.xml*. It runs all UI tests against Chrome and Firefox in parallel.
Note*:* Firefox is run in headless mode.
b. *rest-api-testng.xml.* It runs Rest API tests in parallel.

3. Run TestNG configuration files via Maven.

Run command *mvn install* or *mvn clean test*. It runs suites *crossbrowser-testng.xml*

WARNING!

There are ~20 failed tests for UI and RestAPI. Fails are caused by Integer Overflow. I decided to keep this tests in case of developers fix this issue (joke).

Run JMeter tests

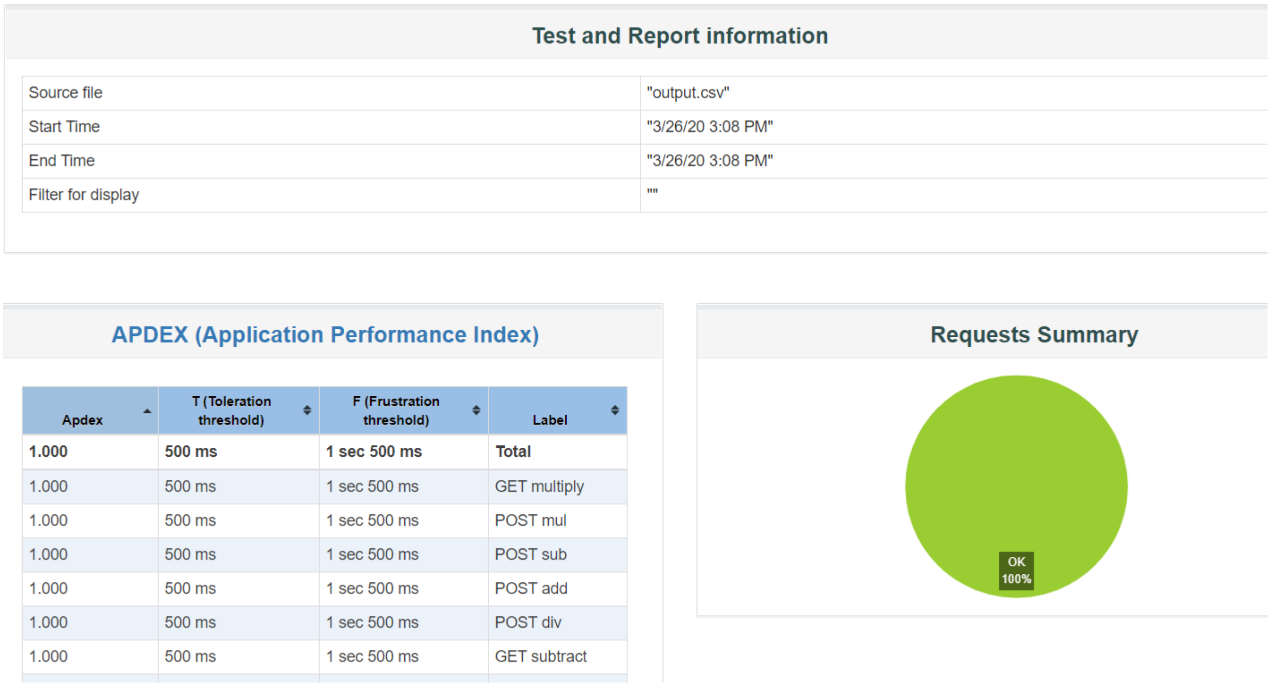
Go to *test-jmeter* folder, launch run.cmd file (for Windows). The report is created after run.

Or just launch JMeter and open .jmx file.

Test reporting

SureFire report created after test run via Maven. It can be found here:
test-automation\target\surefire-reports

JMeter report can be found here:
test-jmeter\reports\<timestamp>



Test and Report information

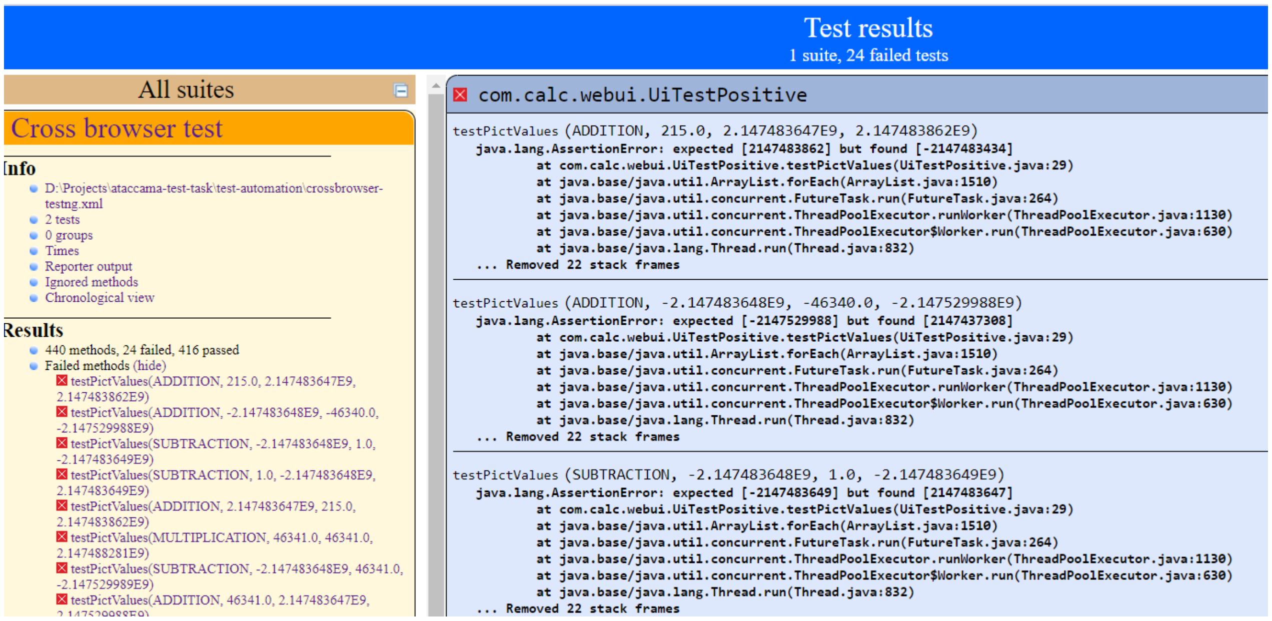
Source file	"output.csv"
Start Time	"3/26/20 3:08 PM"
End Time	"3/26/20 3:08 PM"
Filter for display	""

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**500 ms**	**1 sec 500 ms**	**Total**
1.000	500 ms	1 sec 500 ms	GET multiply
1.000	500 ms	1 sec 500 ms	POST mul
1.000	500 ms	1 sec 500 ms	POST sub
1.000	500 ms	1 sec 500 ms	POST add
1.000	500 ms	1 sec 500 ms	POST div
1.000	500 ms	1 sec 500 ms	GET subtract

Requests Summary

Fig 2.4.1 JMeter report



Test results

1 suite, 24 failed tests

All suites

Cross browser test

Info

- D:\Projects\ataccama-test-task\test-automation\crossbrowser-testng.xml
- 2 tests
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results

- 440 methods, 24 failed, 416 passed
- Failed methods (hide)
  - testPictValues(ADDITION, 215.0, 2.147483647E9, 2.147483862E9)
  - testPictValues(ADDITION, -2.147483648E9, -46340.0, -2.147529988E9)
  - testPictValues(SUBTRACTION, -2.147483648E9, 1.0, -2.147483649E9)
  - testPictValues(SUBTRACTION, 1.0, -2.147483648E9, 2.147483649E9)
  - testPictValues(ADDITION, 2.147483647E9, 215.0, 2.147483862E9)
  - testPictValues(MULTIPLICATION, 46341.0, 46341.0, 2.147488281E9)
  - testPictValues(SUBTRACTION, -2.147483648E9, 46341.0, -2.147529989E9)
  - testPictValues(ADDITION, 46341.0, 2.147483647E9,

com.calc.webui.UiTestPositive

```
testPictValues (ADDITION, 215.0, 2.147483647E9, 2.147483862E9)
   java.lang.AssertionError: expected [2147483862] but found [-2147483434]
           at com.calc.webui.UiTestPositive.testPictValues(UiTestPositive.java:29)
           at java.base/java.util.ArrayList.forEach(ArrayList.java:1510)
           at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
           at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
           at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:630)
           at java.base/java.lang.Thread.run(Thread.java:832)
   ... Removed 22 stack frames

testPictValues (ADDITION, -2.147483648E9, -46340.0, -2.147529988E9)
   java.lang.AssertionError: expected [-2147529988] but found [2147437308]
           at com.calc.webui.UiTestPositive.testPictValues(UiTestPositive.java:29)
           at java.base/java.util.ArrayList.forEach(ArrayList.java:1510)
           at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
           at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
           at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:630)
           at java.base/java.lang.Thread.run(Thread.java:832)
   ... Removed 22 stack frames

testPictValues (SUBTRACTION, -2.147483648E9, 1.0, -2.147483649E9)
   java.lang.AssertionError: expected [-2147483649] but found [2147483647]
           at com.calc.webui.UiTestPositive.testPictValues(UiTestPositive.java:29)
           at java.base/java.util.ArrayList.forEach(ArrayList.java:1510)
           at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
           at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
           at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:630)
           at java.base/java.lang.Thread.run(Thread.java:832)
   ... Removed 22 stack frames
```

Fig 2.4.2. SureFire report

Conclusion

In addition, there can (and should) be added functionality for screenshots capturing (at least for failed scenarios) and log collecting. For more informative reports can be used such tool as Allure Reports. CI can be set up with the help of Jenkins.

But, in my opinion, these additions are out of this test task bounds.