

Testing web applications using Gwen and Selenium

- [Overview](#)
- [Requirements](#)
- [Description](#)
 - [Using Jira and Xray as master](#)
 - [Using Git or other VCS as master](#)
- [References](#)

Overview

In this tutorial, we will perform some web/UI-based tests using [Gwen](#).

Gwen uses the *Given*, *When*, *Then* syntax from Gherkin (thus, its name) to implement an interpretation engine that allows users to easily write "automated tests" (i.e. automated scripts), whose steps will be executed implicitly by their corresponding code implementation. Thus, users can focus on writing (executable) specifications without having to do all the implementation hard-work.

Gwen also separates declarative from imperative style Gherkin specifications. Declarative is done in standard `.feature` files that may include steps defined in while [imperative specifications](#) (i.e. "meta-features") are managed in `.meta` files.

Gwen uses Selenium under the hood, by providing a DSL that allows users to interact with the browser without having to write code.

From the many interesting features of Gwen we can highlight the auto-update capability and also the ability taking screenshots, which will be available for analysis after tests are run.

Requirements

- gwen
- gwen-web
- cucumber-json-merge
 - `npm install -g cucumber-json-merge`

Description

We will use sample code from [gwen-web repository](#), using some [instructions](#) available online.

Remember that we need to manage:

- features (declarative specifications, usually stored in `.feature` files)
- meta-features (imperative specifications, usually stored in `.meta` files)

Besides that, you need to decide is which workflow we'll use: do we want to use Xray/Jira as the master for writing the declarative specification or do we want to manage those in Git?



Learn more

Please see [Testing in BDD with Gherkin based frameworks \(e.g. Cucumber\)](#) for an overview of the possible workflows.

Using Jira and Xray as master

This section assumes using Xray as master, i.e. the place that you'll be using to edit the specifications (e.g. the scenarios that are part of `.feature` files).



Please note

This tutorial explores using Xray for storing and managing the declarative scenarios and not the ones contained within the meta-features.

However, it should also be possible to manage them as Test issues with a "StepDef" label; it would require further evaluation though.

The first step is to create a Cucumber Test, of Cucumber Type "Scenario", in Jira. The specification would be exactly the same as the one provided in the original repository.

The test is quite self-explanatory, which is the ultimate purpose of using this approach: a browser is open, then we search by “Gwen automation” and then we look at the first Google result.

Calculator / CALC-4823

Perform a google search

Test Details

Type: Cucumber

Scenario Type: Scenario

Scenario:

1 Given I have Google in my browser

2 When I do a search for "Gwen automation"

3 Then the first result should open a Gwen page

Autocomplete based on labels: Filter Labels

Press

Ctrl

 +

Space

 to get step suggestions.

Save

Cancel

After creating the Test in Jira and associating it with requirements, etc., you can export the specification of the test to a Cucumber .feature file via the REST API, or the **Export to Cucumber** UI action from within the Test/Test Execution issue or even based on an existing saved filter. A plugin for your CI tool of choice can be used to ease this task.

Calculator / CALC-4823

Perform a google search

Edit

Comment

Assign

More

Start Progress

Resolve Issue

Close Issue

Admin

Details

Type: Test

Affects Version/s: None

Component/s: None

Labels: None

Description

Click to add description

Test Details

Type: Cucumber

Scenario Type: Scenario

Scenario: Given I have Google in my browser
When I do a search for "Gwen automation"
Then the first result should open a Gwen page

Pre-Conditions

Log work

Agile Board

Rank to Top

Rank to Bottom

Attach files

Voters

Stop watching

Watchers

Create sub-task

Convert to sub-task

Move

Link

Clone

Labels

Assess risk

Delete

Reset TestRunStatus

Export to Cucumber

Status: OPEN (View Workflow)

Resolution: Unresolved

Fix Version/s: None

Edit Steps

The coverage and the test results can be tracked in the "requirement" side (e.g. user story).

Calculator / CALC-4805

Google search (gwen-web-demo)

EditCommentAssignMore

Start ProgressResolve IssueClose IssueAdmin

Details

Type:Story

Priority:Major

Affects Version/s:None

Component/s:None

Labels:None

Requirement Status:v3.0 - NOTRUN

Status:OPEN (View Workflow)

Resolution:Unresolved

Fix Version/s:None

Description

Click to add description

Test Coverage

Create TestCreate Sub-Test ExecutionLink Tests

Versionv3.0

All EnvironmentsNOT RUN

OPENUnresolvedCALC-4823Perform a google searchTODO

After being exported, the created .feature file will be similar to the original but will contain the references to the Test issue key and the covered requirement issue key.

```
features/google.feature

@REQ_CALC-4805
Feature: Google search (gwen-web-demo)

    @TEST_CALC-4823
    Scenario: Perform a google search
        Given I have Google in my browser
        When I do a search for "Gwen automation"
        Then the first result should open a Gwen page
```

The steps correspond to reusable blocks, defined as @StepDef scenarios within meta-feature files like the following one. This is the automation glue.

meta/google/Google.meta

Feature: Google search meta

@StepDef

Scenario: I have Google in my browser

Given I start a new browser

When I navigate to "http://www.google.com"

Then the page title should be "Google"

@StepDef

Scenario: I do a search for "<query>"

Given the search field can be located by name "q"

When I enter "<query>" in the search field

Then the page title should contain "<query>"

@StepDef

Scenario: the first result should open a Gwen page

Given the first match can be located by css selector ".r > a"

When I click the first match

Then the current URL should contain "gwen-interpreter"

In this example, we're assuming that this meta-feature is not imported to Xray nor managed there; thus, it will probably live in the VCS.

Besides the previous example, there are also [additional tests](#) for interacting with a demo page, with corresponding [meta specification](#).

Gwen loads both standard and meta-features and finds the right code to execute.

After running the tests and generating the Cucumber JSON report (e.g., [merged-test-results.json](#)), it can be imported to Xray via the REST API or the **Import Execution Results** action within the Test Execution.

The [cucumber-json-merge](#) utility may be handy to merge the results of each feature.

```
./gwen -b -m meta -f json -r target/reports features
cucumber-json-merge -d target/reports/json/
curl -H "Content-Type: application/json" -X POST -u admin:admin --data @"merged-test-results.json"
http://jiraserver.example.com/rest/raven/1.0/import/execution/cucumber
```



Calculator / CALC-5152

Execution results [1572251627992]

[Edit](#) [Comment](#) [Assign](#) [More](#) [Close Issue](#) [Reopen Issue](#) [Admin](#)

Details

Type: **Test Execution** Status: **RESOLVED** [\(View Workflow\)](#)
Affects Version/s: **None** Resolution: **Fixed**
Component/s: **None** Fix Version/s: **None**
Labels: **None**
Test Environments: **None**
Test Plan: **None**

Description

Execution results imported from external source

Tests

[+ Add](#)

Overall Execution Status

8 PASS **1** FAIL

TOTAL TESTS: 9

[Filter\(s\)](#)



[Apply Rank](#)

Show **100** entries

[Columns](#)

	Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
<input type="checkbox"/>	1	CALC-4823	Perform a google search	Cucumber	1	0	Administrator	FAIL	▶ ...
<input type="checkbox"/>	2	CALC-4824	Initialise user agent	Cucumber	1	0	Administrator	PASS	▶ ...
<input type="checkbox"/>	3	CALC-4825	Launch the challenge	Cucumber	1	0	Administrator	PASS	▶ ...
<input type="checkbox"/>	4	CALC-4826	Complete step 1	Cucumber	1	0	Administrator	PASS	▶ ...
<input type="checkbox"/>	5	CALC-4827	Complete step 2	Cucumber	1	0	Administrator	PASS	▶ ...
<input type="checkbox"/>	6	CALC-4828	Complete step 3	Cucumber	1	0	Administrator	PASS	▶ ...

The execution screen details will provide information on the test run result that includes step-level information including duration.

Calculator / Test Execution: CALC-5152 / Test: CALC-4823

Perform a google search



[Import Execution Results](#)

[Export to Cucumber](#)

[Return to Test Execution](#)

[Next](#)

tests

[CALC-4805](#) Google search (gwen-web-demo)



[OPEN](#)

Test Details

Test Type: **Cucumber**
Scenario Type: **Scenario**
Scenario:

- Given I have Google in my browser
- When I do a search for "Gwen automation"
- Then the first result should open a Gwen page

Results

Context	Duration	Status
-	7 sec	FAIL
Steps		
Given I have Google in my browser	1673.623 ms	PASS
When I do a search for "Gwen automation"	4138.371 ms	PASS
Then the first result should open a Gwen page	1766.212 ms	FAIL
Failed step [at line 19]: Then the current URL should contain "gwen-interpreter": assertion failed: Expected the current URL to contain 'gwen-interpreter' but got 'https://www.gwen.com/'		

[evidence_step_3_0.png](#)

[evidence_step_3_1.png](#)

As shown above, besides a detailed error message, screenshots are also automatically available on failed steps.

On the "requirement"/user story side (i.e the "feature") we can also see how this result impacting on the coverage.



Calculator / CALC-4805

Google search (gwen-web-demo)

[Edit](#) [Comment](#) [Assign](#) [More ▾](#) [Start Progress](#) [Resolve Issue](#) [Close Issue](#) [Admin ▾](#)

Details

Type: [Story](#) Status: [OPEN](#) ([View Workflow](#))
Priority: [Major](#) Resolution: [Unresolved](#)
Affects Version/s: [None](#) Fix Version/s: [None](#)
Component/s: [None](#)
Labels: [None](#)
Requirement Status: [v3.0 - NOTRUN](#)

Description

[Click to add description](#)

Test Coverage

[Create Test](#) [Create Sub-Test Execution](#) [Link Tests](#)

Version ▾ [None - latest execution](#) ▾ All Environments ▾ [NOK](#) [←](#)

[OPEN](#) [Unresolved](#) [CALC-4823](#) [Perform a google search](#) [→](#) [FAIL](#)

If we wanted to correct the previous error, in this case, we would need to correct the meta-feature file containing the specification of the step “Then the first result page should open a Gwen page” and run the tests again.

Calculator / Test Execution: CALC-5154 / Test: CALC-4823

Perform a google search

[Import Execution Results](#) [Export to Cucumber](#) [Return to Test Execution](#) [Next ▸](#)

Test Issue Links (1)

tests [CALC-4805](#) Google search (gwen-web-demo) [↑](#) [OPEN](#)

Test Details

Test Type: [Cucumber](#)
Scenario Type: [Scenario](#)
Scenario:

- 1 [Given](#) I have Google in my browser
- 2 [When](#) I do a search for "Gwen automation"
- 3 [Then](#) the first result should open a Gwen page

Results

Context	Duration	Status
Steps	10 sec	PASS
Given I have Google in my browser	6034.178 ms	PASS
When I do a search for "Gwen automation"	2044.482 ms	PASS
Then the first result should open a Gwen page	2800.294 ms	PASS

Using Git or other VCS as master

You can edit your .feature and .meta files outside of Jira (eventually storing them in your VCS using Git, for example).

In any case, you'll need to synchronize your .feature files to Jira so that you can have visibility of them and report results against them.

Thus, you need to import your .feature files to Xray/Jira; you can invoke the REST API directly or use one of the available plugins/tutorials for CI tools.

```
zip -r features.zip features/ -i \*.feature
curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@features.zip" "http://jiraserver.example.com/rest/raven/1.0/import/feature?projectKey=CALC"
```



Please note

Each Scenario of each .feature will be created as a Test issue that contains unique identifiers, so that if you import once again then Xray can update the existent Test and don't create any duplicated tests.

Afterward, you can export those features out of Jira based on some criteria, so they are properly tagged, run them and import back the results to correct entities in Xray.

References

- <https://gwen-interpreter.github.io/>
- <https://github.com/gwen-interpreter/gwen>
- <https://github.com/gwen-interpreter/gwen-web>
- <https://gweninterpreter.wordpress.com/>
- [Testing in BDD with Gherkin based frameworks \(e.g. Cucumber\)](#)
- <https://github.com/bitcoder/cucumber-json-merge>
- [Automated Tests \(Import/Export\)](#)
- [Exporting Cucumber Tests - REST](#)