

# Testing web applications using Gwen and Selenium

- [Overview](#)
- [Requirements](#)
- [Description](#)
  - [Using Jira and Xray as master](#)
  - [Using Git or other VCS as master](#)
- [FAQ and Recommendations](#)
- [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-28299

Perform a google search

Attach

Create subtask

Link issue

Link page

Description

Add a description...

Linked issues

tests

CALC-28317

Google search (gwen-web-demo)

↑

TO DO

Test Details

Cucumber

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

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-28299

Perform a google search

Attach

Create subtask

Link issue

Link page

Description

Add a description...

Test Details

Cucumber

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

9 of 9

STATUS

To Do

ASSIGNEE

Unassigned

REPORTER

AR André Miguel Pereira Rodrig

LABELS

None

CENAS

None

BEGIN DATE

Log work

Add flag

Xporter for JIRA

Export to Cucumber

Convert to Subtask

Move

Clone

Delete

NEW JIRA ISSUE VIEW

Turn off new issue view

Configure

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

CALC-28317

## Google search (gwen-web-demo)

Attach Create subtask Link issue Link page

### Description

Add a description...

### Linked issues

is tested by

CALC-28299 Perform a google search

### Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test

Latest Version Test Plan

Test Environment

All Environments

NOTRUN

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ TO DO	CALC-28299	Perform a google search	TO DO

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-28317
Feature: Google search (gwen-web-demo)
```

```
  @TEST_CALC-28299
  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/

# submit from the command line
BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer $token" --data @"merged-test-results.json" "$BASE_URL/api/v2/import/execution/cucumber"
```

▶ CALC-28318

## Execution results [1574163423567]

[Attach](#) [Create subtask](#) [Link issue](#) [Link page](#) [Open](#) [...](#)

### Description

Add a description...

### Tests

Create Test

+ Add ▾

### Overall Execution Status

TOTAL TESTS: 9

8 PASSED 1 FAILED

	Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/>	1	<a href="#">CALC-28299</a>	Perform a google search	Cucumber	FAILED	<a href="#">View</a> <a href="#">...</a>
<input type="checkbox"/>	2	<a href="#">CALC-28300</a>	Complete step 1	Cucumber	PASSED	<a href="#">View</a> <a href="#">...</a>
<input type="checkbox"/>	3	<a href="#">CALC-28301</a>	Complete step 4	Cucumber	PASSED	<a href="#">View</a> <a href="#">...</a>

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

Calculator / Test Execution: CALC-28318 / Test: CALC-28299

[Perform a google search](#)

[Import Execution Results](#)

[Export to Cucumber](#)

[Return to Test Execution](#)

[Next](#)

Execution Status **FAILED**



Started On: 19/Nov/2019 11:37 AM

Finished On: 19/Nov/2019 11:37 AM

Assignee:

[Sérgio Freire](#)

Versions:

Revision:

Executed By:

[Sérgio Freire](#)

Test Environments:

Comment

[Preview comment](#)

Execution Defects (0)

+ ▾

Execution Evidence (0)

[Add Evidence](#)

## Execution Details

### Test Description

### Test Issue Links (1)

tests [CALC-28317](#) Google search (gwen-web-demo)

### 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	8 secs	FAILED
Given I have Google in my browser	1 secs	PASSED
When I do a search for "Gwen automation"	1 secs	PASSED
Then the first result should open a Gwen page	5 secs	FAILED
Failed step [at line 18]: When I click the first match: Could not locate element: the first match		

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.

■ CALC-28317

## Google search (gwen-web-demo)

 Attach  Create subtask  Link issue  Link page  

### Description

Add a description...

### Linked issues

is tested by

■ CALC-28299 Perform a google search  **TO DO**

### Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution

Create new Test



**Latest** Version Test Plan

Test Environment

All Environments 

**NOK**

 Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
 <b>TO DO</b>	<b>CALC-28299</b>	Perform a google search	 <b>FAILED</b>

Prev **1** Next

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.

## 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
BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: multipart/form-data" -H "Authorization: Bearer $token" -F "file=@features.zip" "$BASE_URL/api/v2/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.

If we change the specification (i.e. the Gherkin scenarios), we need to import the .feature(s) once again.

Therefore, in the CI we always need to start by importing the .feature file(s) to keep Jira/Xray on synch.

## FAQ and Recommendations

Please see [this page](#).

## 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