Testing web applications using Gwen and Selenium

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.

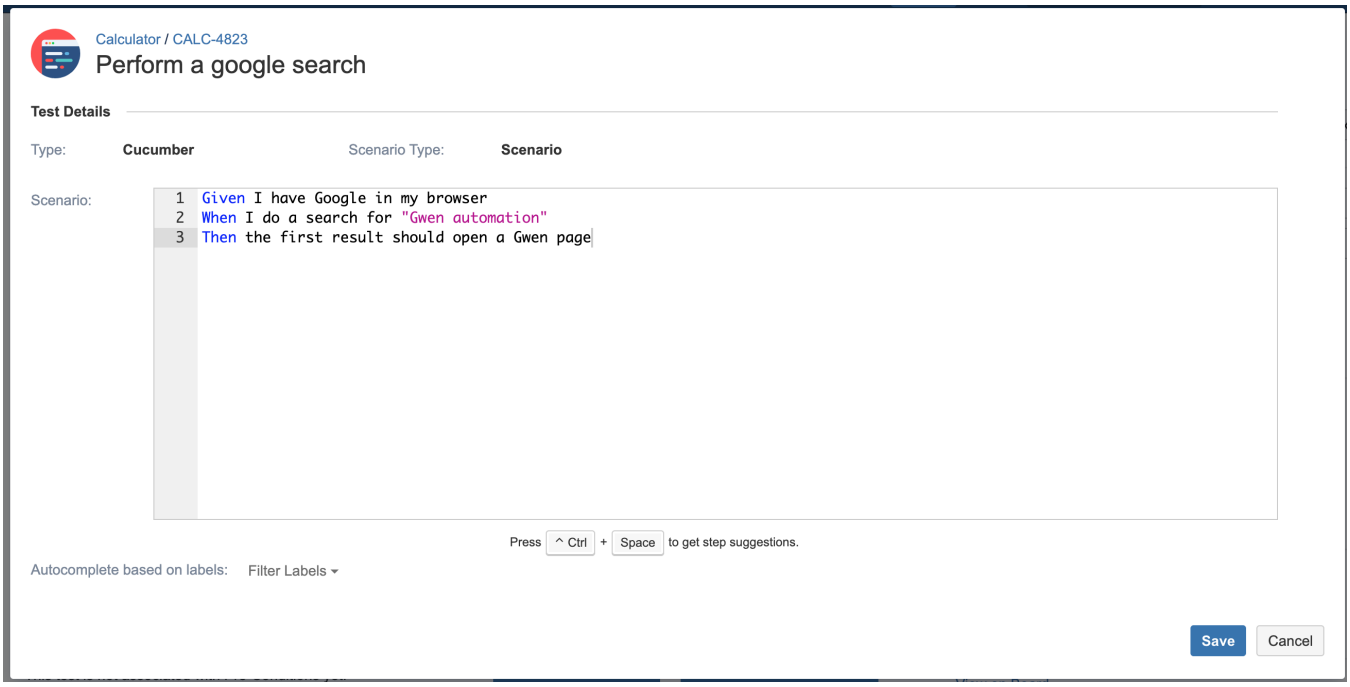
Using Jira and Xray as master

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).

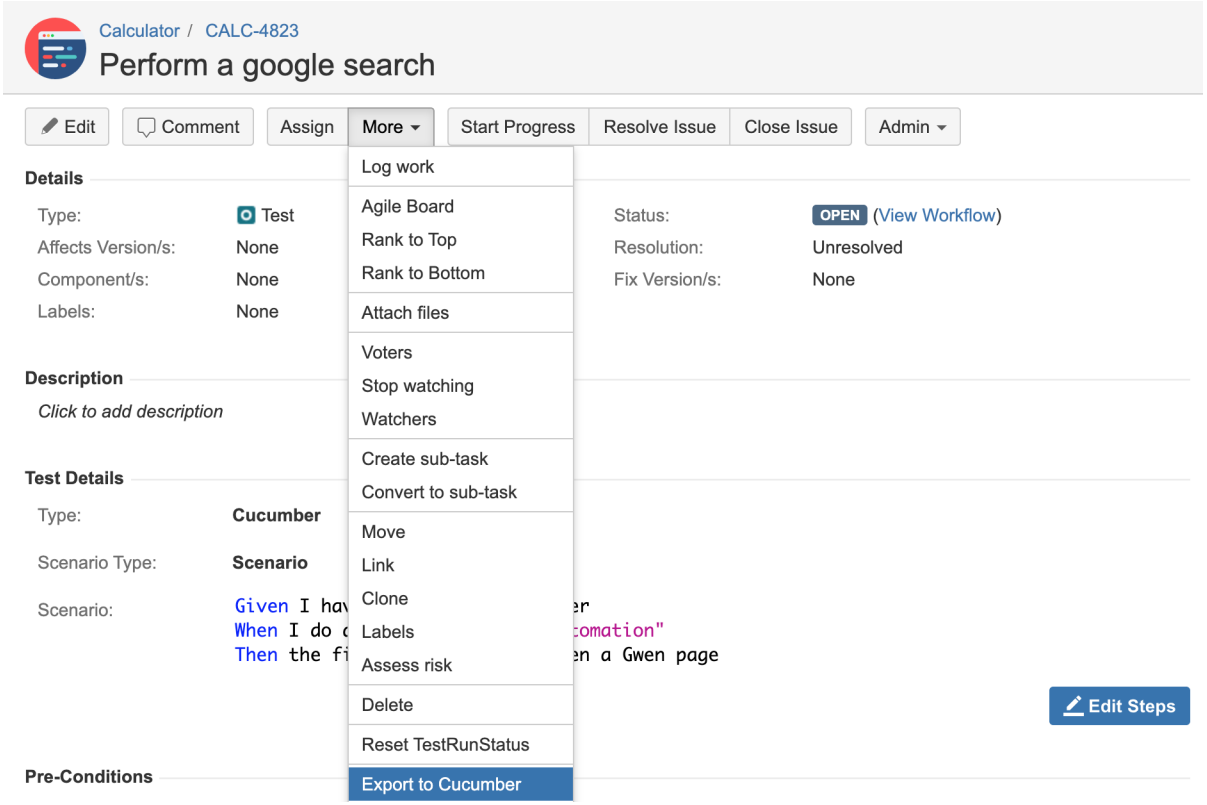
> **Please note**
>
> This tutorial explores using Xray for storing and managing the declarative scenarios and not the ones contained within the meta-features.
>
> However, it should also be possible to manage them as Test issues with a "StepDef" label; it would require further evaluation though.

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.

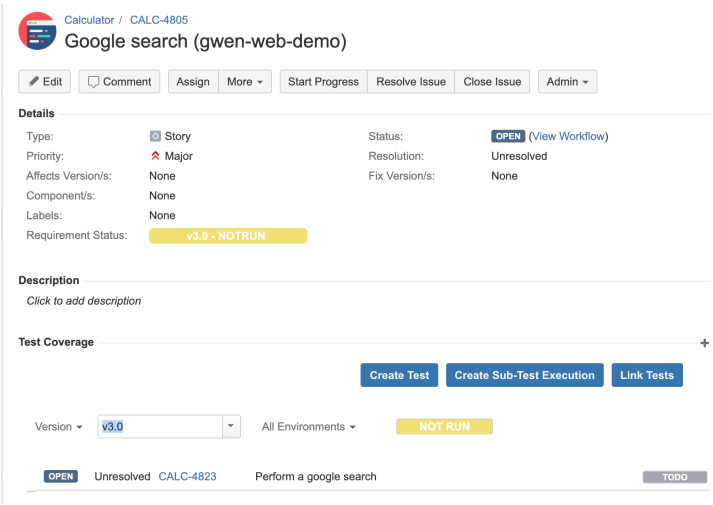
The test is quite self-explanatory, which is the ultimate purpose of using this approach: a browser is open, then we search by “Gwen automation” and then we look at the first Google result.



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.



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



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.

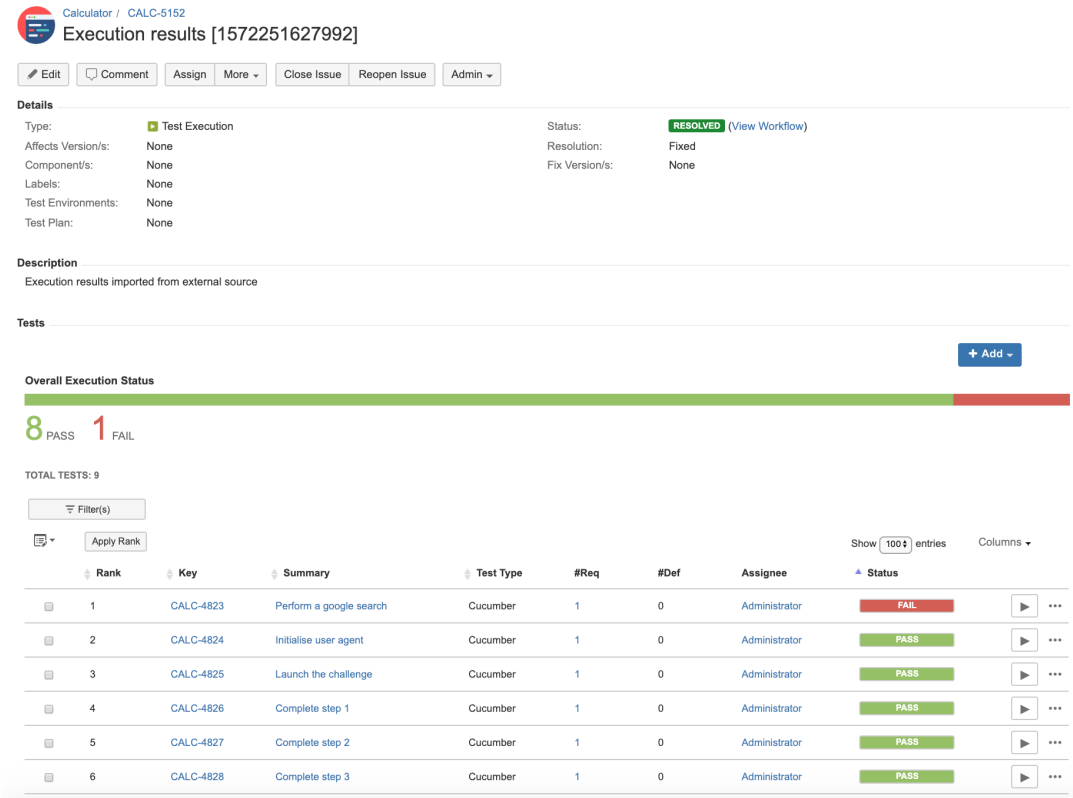
Besides the previous example, there are also additional tests for interacting with a demo page, with corresponding meta specification.

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

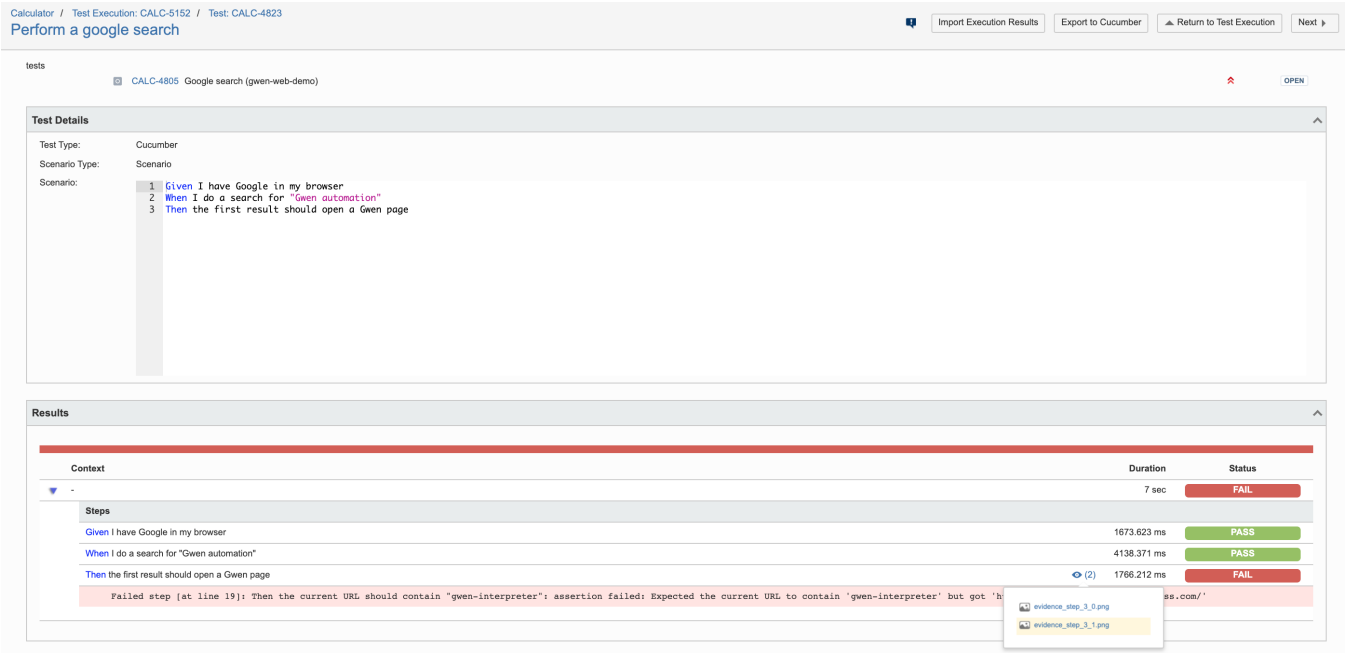
After running the tests and generating the Cucumber JSON report (e.g., merged-test-results.json), it can be imported to Xray via the REST API or the **Import Execution Results** action within the Test Execution.

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
```

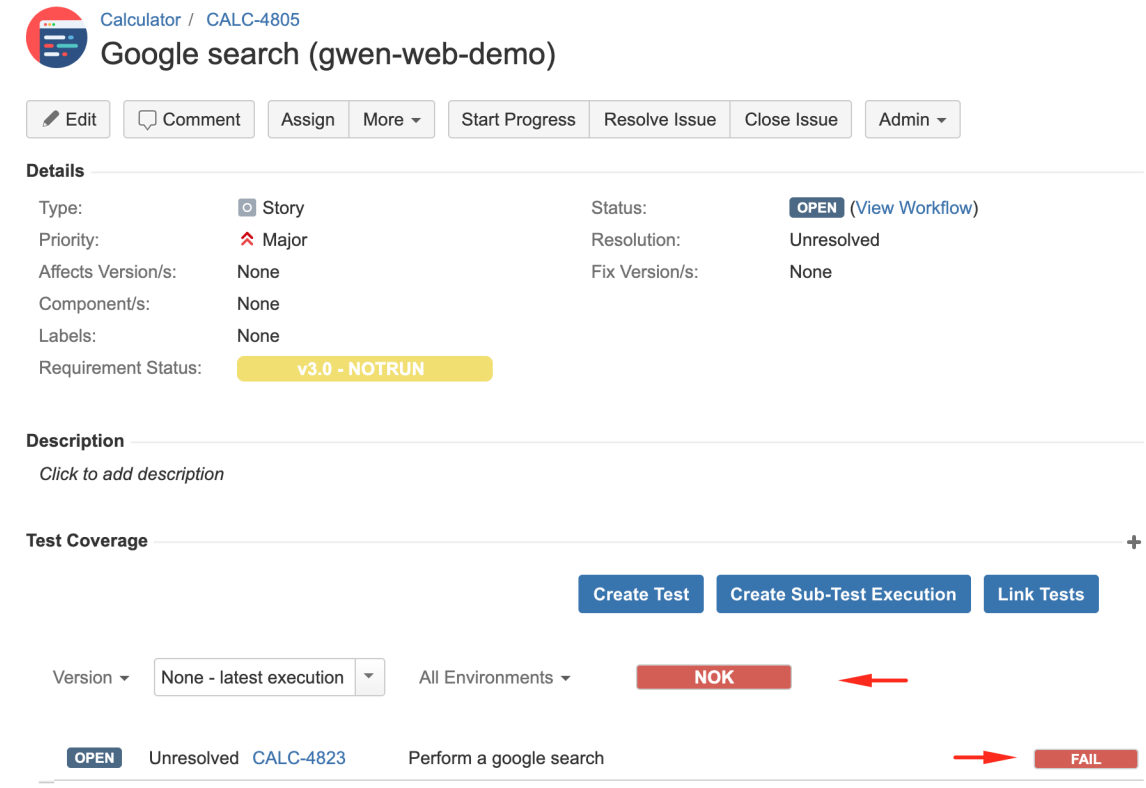


The execution screen details will provide information on the test run result that includes step-level information including duration.

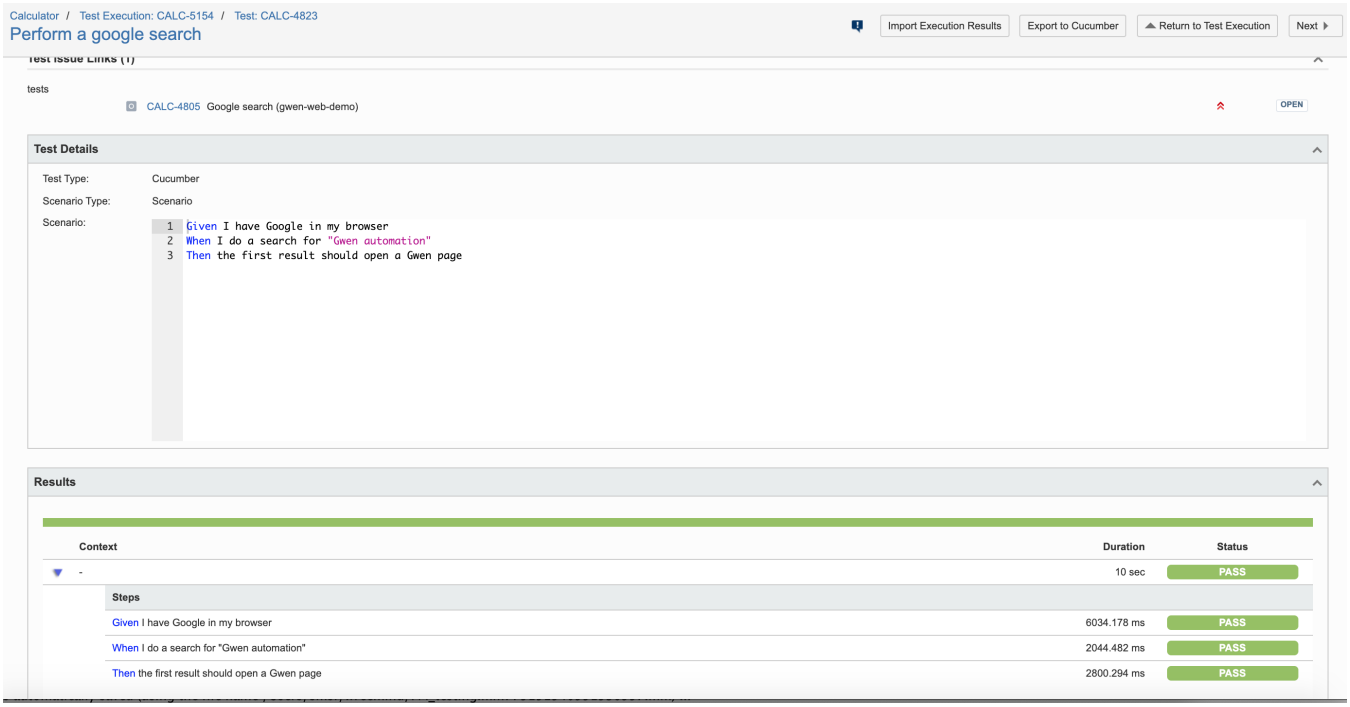


As shown above, besides a detailed error message, screenshots are also automatically available on failed steps.

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



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
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