Testing Node.js apps using Cucumber.js in JavaScript

- Overview
- Requirements
- Description
    - Using Jira and Xray as master
- References

Overview

In this tutorial, we will create tests for Node.js, in JavaScript, with Cucumber.js.

The test (specification) is initially created in Jira as a Cucumber Test and afterwards, it is exported using the UI or the REST API.

Requirements

- nodejs
- npm packages
    - cucumber

Description

For the purpose of this tutorial, we'll use a simple JavaScript class implementing a very basic calculator.

lib/calculator.js

```
class Calculator {

    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    add() {
        this.result = this.x + this.y;
    }

    getResult() {
        return this.result;
    }
}

module.exports = Calculator;
```

We aim to test the sum operation.

However, before moving into the actual implementation, you need to decide which workflow to use: do you want to use Xray/Jira as the master for writing the declarative specification, or do you want to manage those in Git?

This tutorial only showcases how to use Xray/Jira as the master for editing the Cucumber Scenarios/Scenario Outlines.

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).

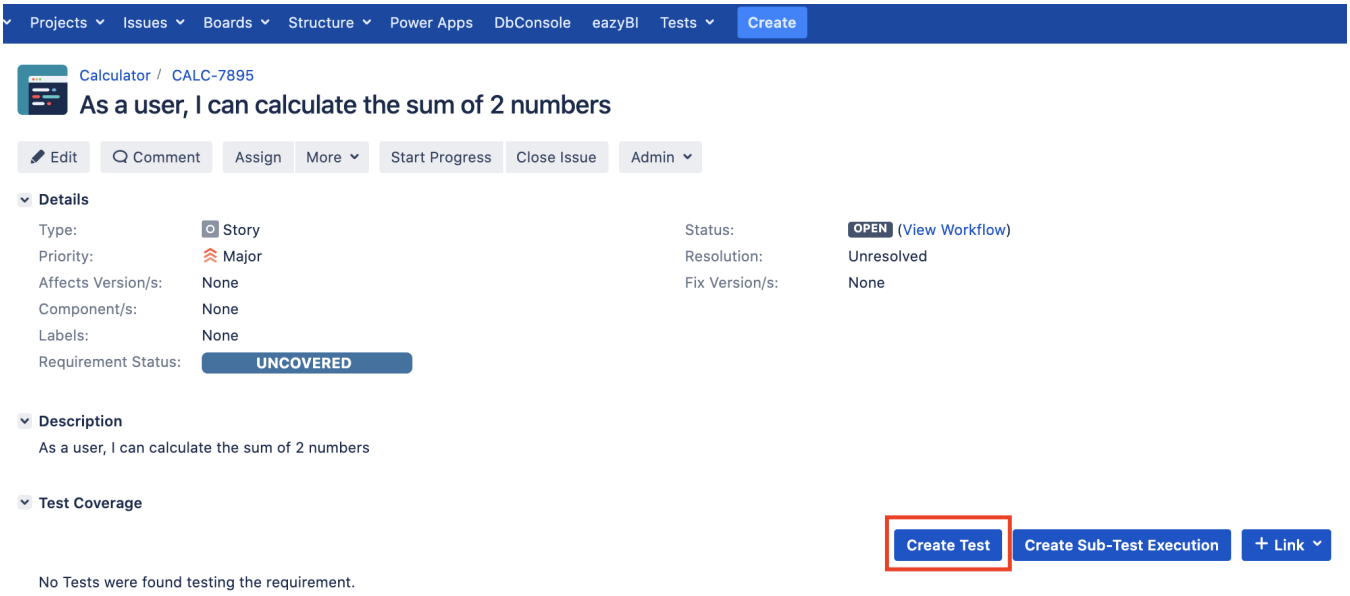
The overall flow would be something like this:

1. create Scenario/Scenario Outline as a Test in Jira; usually, it would be linked to an existing "requirement"/Story (i.e. created from the respective issue screen)
2. implement the code related to Gherkin statements/steps and store it in Git, for example
3. generate .feature files based on the specification made in Jira
4. checkout the code from Git
5. run the tests in the CI
6. import the results back to Jira

Usually, you would start by having a Story, or similar (e.g. "requirement"), to describe the behavior of a certain feature and use that to drive your testing.

If you have it, then you can just use the "Create Test" on that issue to create the Scenario/Scenario Outline and have it automatically linked back to the Story/"requirement."

Otherwise, you can create the Test using the standard (issue) Create action from Jira's top menu.

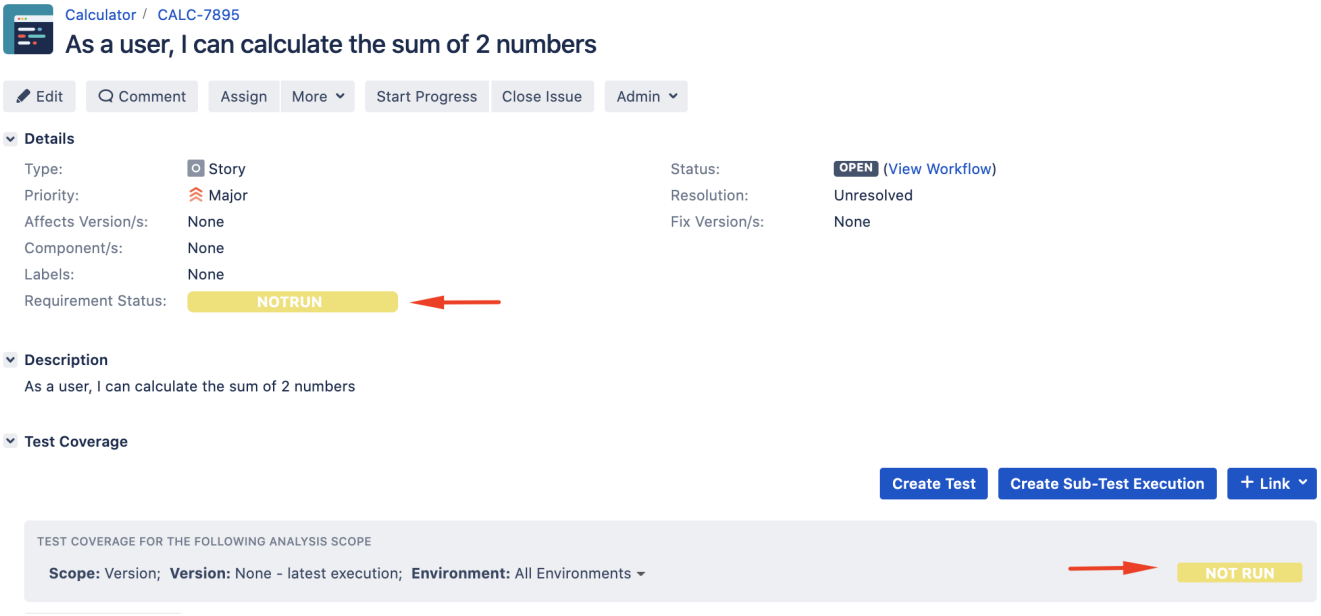


In this case, we'll create a Cucumber Test, of Cucumber Type "Scenario."

We can fill out the Gherkin statements immediately on the Jira issue create dialog or we can create the Test issue first and fill out the details on the next screen, from within the Test issue. In the latter case, we can take advantage of the built-in Gherkin editor which provides auto-complete of Gherkin steps.

After the Test is created it will impact the coverage of related "requirement," if any.

The coverage and the test results can be tracked in the "requirement" side (e.g. user story). In this case, it changed from being UNCOVERED to NOTRUN (i.e. covered and with at least one test not run).



The related statement's code is managed outside of Jira and stored in Git, for example.

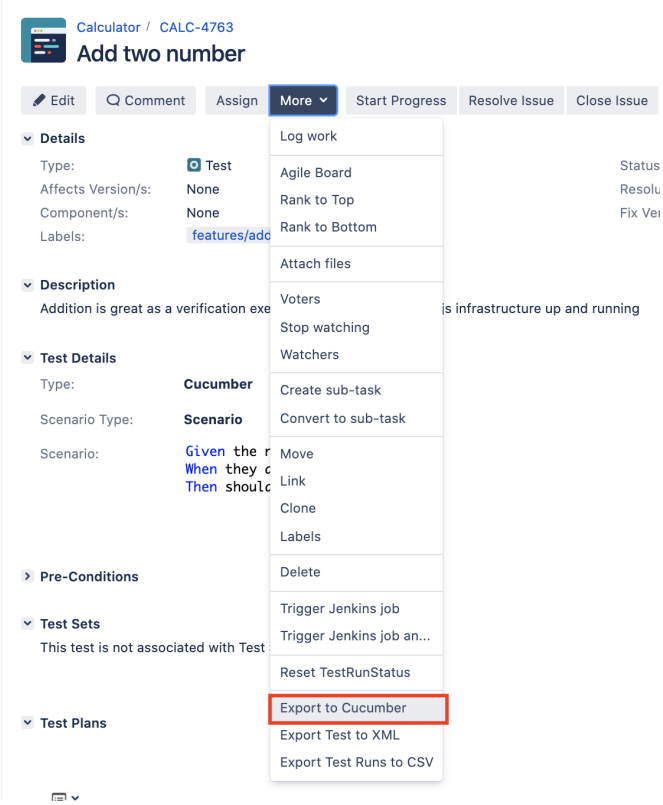
features/step_definitions/addition_steps.js

```
const assert = require('assert')
const {Before, Given, When, Then} = require('cucumber');
const Calculator = require('../../lib/calculator');

let calculator;

Given('the numbers {int} and {int}', function (x, y) {
    calculator = new Calculator(x, y);
});

When('they are added together', function () {
    calculator.add();
});

Then('should the result be {int}', function (expected) {
    assert.equal(calculator.getResult(), expected)
});
```

You can then 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.

So, you can either:

- use the UI



- use the REST API (more info here)

example of a Bash script to export/generate the features from Xray

```
#!/bin/bash

curl -u admin:admin  "http://jiraserver.example.com/rest/raven/1.0/export/test?keys=CALC-4763&fz=true" -o features.zip
rm -f features/*.feature
unzip -o features.zip  -d features
```

- use one of the available CI/CD plugins (e.g. see an example of Integration with Jenkins)

After being exported, the created .feature(s) will contain references to the Test issue key and the covered "requirement" issue key, if that's the case. The naming of these files is detailed in Export Cucumber Features.

new feature after export (features/1_CALC-7895.feature)

```
@REQ_CALC-7895
Feature: As a user, I can calculate the sum of 2 numbers

        #  Addition is great as a verification exercise to get the Cucumber-js infrastructure up and running
        @TEST_CALC-4763 @features/addition.feature
        Scenario: Add two number
                Given the numbers 2 and 3
                When they are added together
                Then should the result be 5
```

To run the tests and produce a Cucumber JSON report, we can use the `cucumber-js` binary.

```
./node_modules/cucumber/bin/cucumber-js -f json:report.json
```

After running the tests, results can be imported to Xray via the REST API, the **Import Execution Results** action within the Test Execution, or by using one of the available CI/CD plugins (e.g. see an example of Integration with Jenkins).

```
curl -H "Content-Type: application/json" -X POST -u admin:admin --data @"report.json" http://jiraserver.example.com/rest/raven/1.0/import/execution/cucumber
```

Which Cucumber endpoint/"format" to use?

To import results, you can use two different endpoints/"formats" (endpoints described in Import Execution Results - REST):

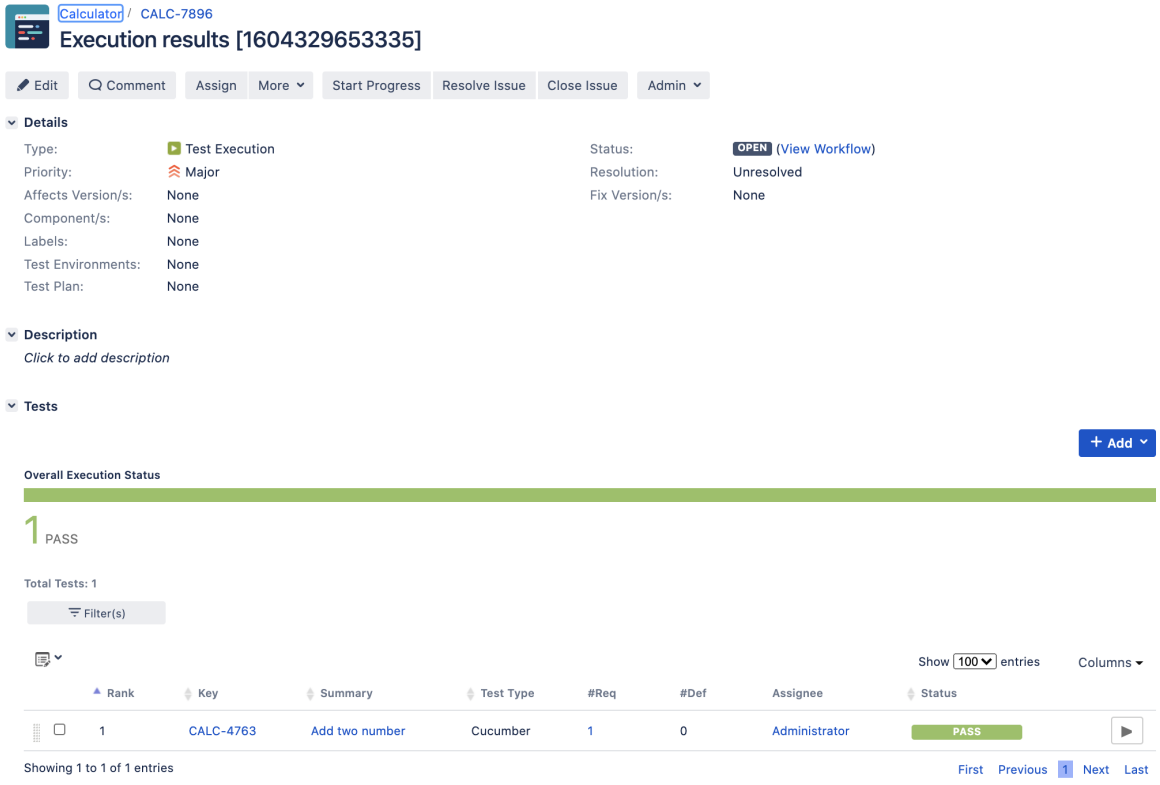
1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

The standard cucumber endpoint (i.e. */import/execution/cucumber)* is simpler but more restrictive: you cannot specify values for custom fields on the Test Execution that will be created. This endpoint creates new Test Execution issues unless the Feature contains a tag having an issue key of an existing Test Execution.

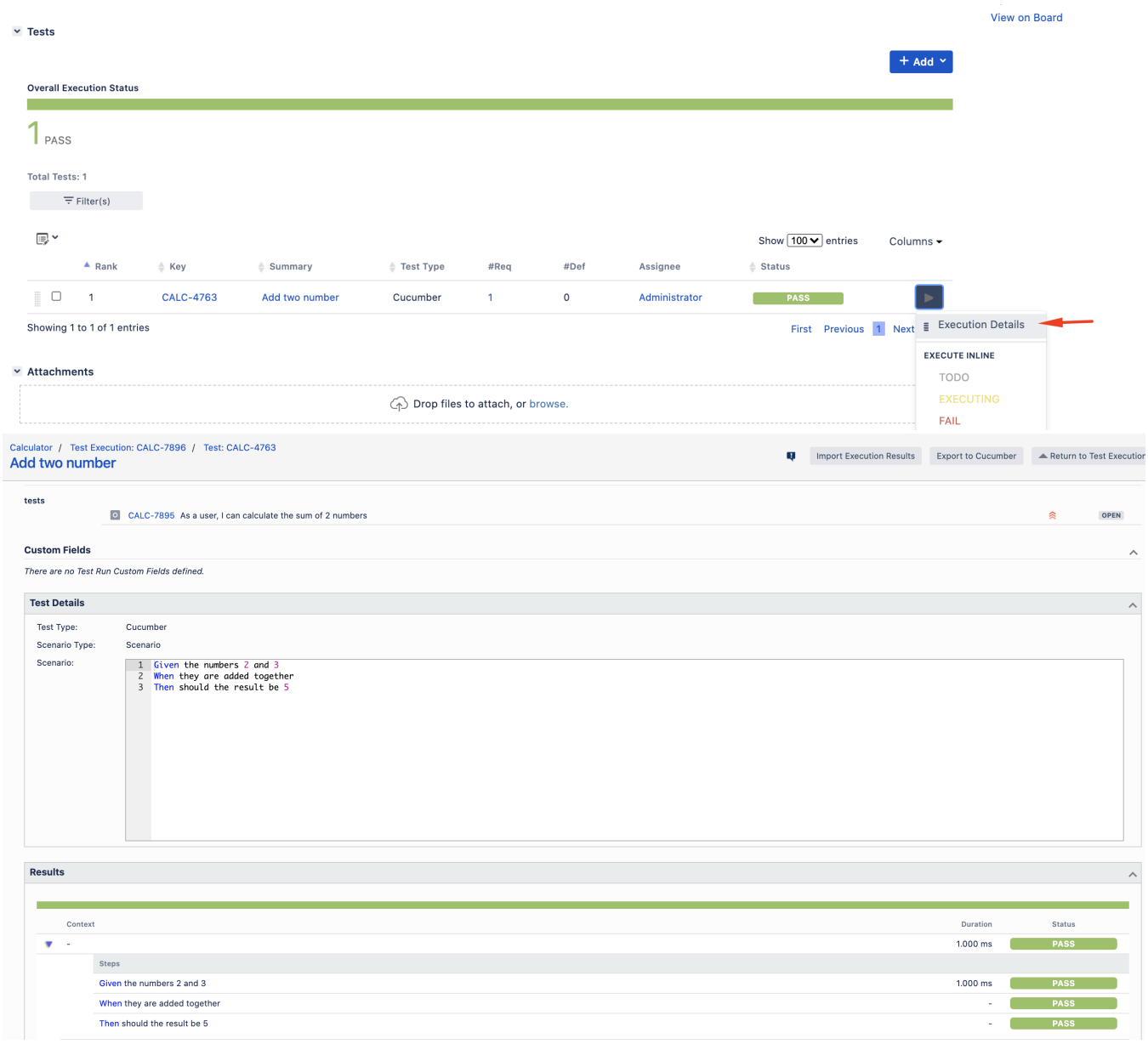
The multipart cucumber endpoint will allow you to customize fields (e.g. Fix Version, Test Plan) if you wish to do so on the Test Execution that will be created. Note that this endpoint always creates new Test Executions (as of Xray v4.2).

In sum, if you want to customize the Fix Version, Test Plan and/or Test Environment of the Test Execution issue that will be created, you'll have to use the "multipart cucumber" endpoint.

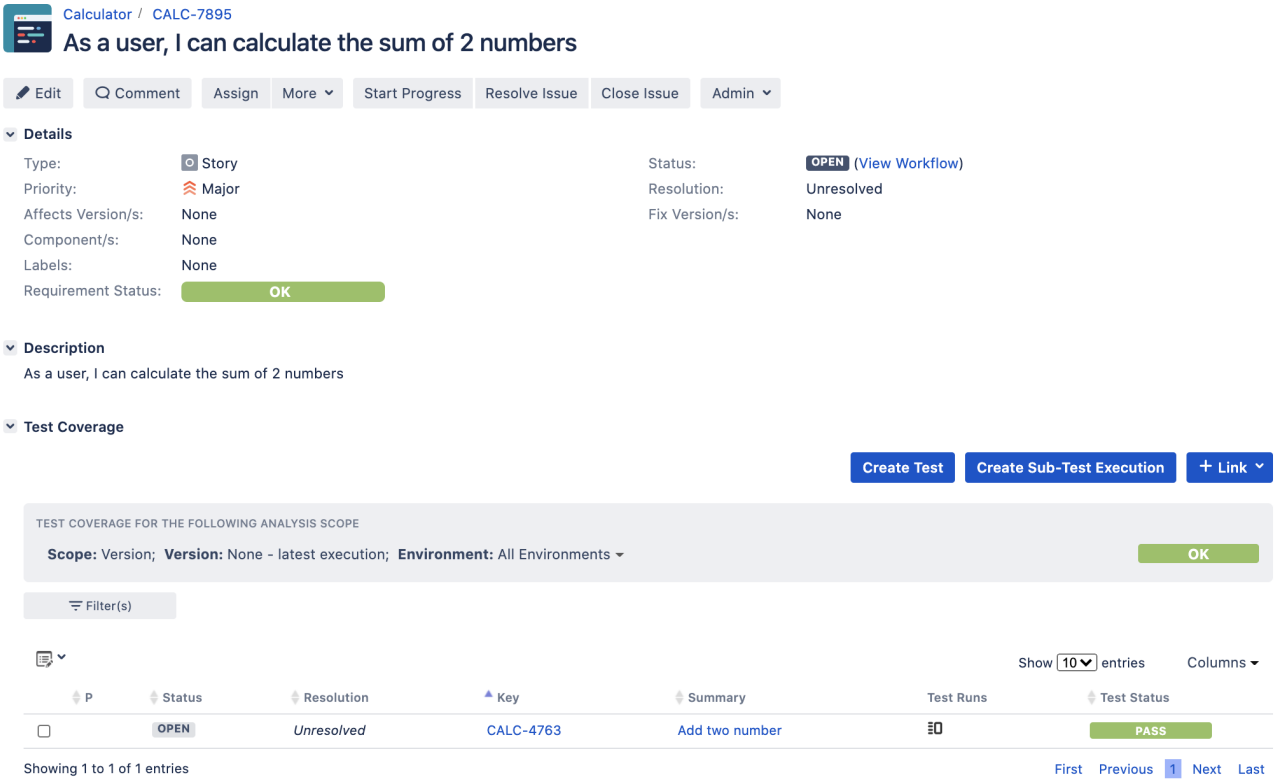
A new Test Execution will be created (unless you originally exported the Scenarios/Scenario Outlines from a Test Execution).



The execution screen details of the Test Run will provide overall status information and Gherkin statement-level results.



Results are reflected on the covered item (e.g. Story). On the issue screen, coverage now shows that the item is OK based on the latest testing results, that can also be tracked within the Test Coverage panel bellow.



References

- https://github.com/cucumber/cucumber-js
- Testing in BDD with Gherkin based frameworks (e.g. Cucumber)
- Automated Tests (Import/Export)
- Exporting Cucumber Tests - REST