Testing using Behat in PHP

Overview

In this tutorial, we will implement some Gherkin tests using Behat and PHP.

Requirements

- behat
- behat-cucumber-formatter (patched)

Description

We will use the example provided in Behat's quick start documentation, which describes a feature of a *product basket*.

The classes implementing our basket are quite simple and reflect the rules defined for the feature:

- VAT is 20%
- Delivery for basket under £10 is £3
- Delivery for basket over £10 is £2

features/bootstrap/Basket.php

```php
<?php

// features/bootstrap/Basket.php

final class Basket implements \Countable
{
    private $shelf;
    private $products;
    private $productsPrice = 0.0;

    public function __construct(Shelf $shelf)
    {
        $this->shelf = $shelf;
    }

    public function addProduct($product)
    {
        $this->products[] = $product;
        $this->productsPrice += $this->shelf->getProductPrice($product);
    }

    public function getTotalPrice()
    {
        return $this->productsPrice
            + ($this->productsPrice * 0.2)
            + ($this->productsPrice > 10 ? 2.0 : 3.0);
    }

    public function count()
    {
        return count($this->products);
    }
}
```

features/bootstrap/Shelf.php

```php
<?php

// features/bootstrap/Shelf.php

final class Shelf
{
    private $priceMap = array();

    public function setProductPrice($product, $price)
    {
        $this->priceMap[$product] = $price;
    }

    public function getProductPrice($product)
    {
        return $this->priceMap[$product];
    }
}
```

We aim to use Gherkin in Behat to describe our scenarios and have an executable specification.

Remember that we need to manage:

- features (declarative specifications, usually stored in .feature files)
- corresponding automation code glue

Besides that, you'll need to decide which workflow to 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.

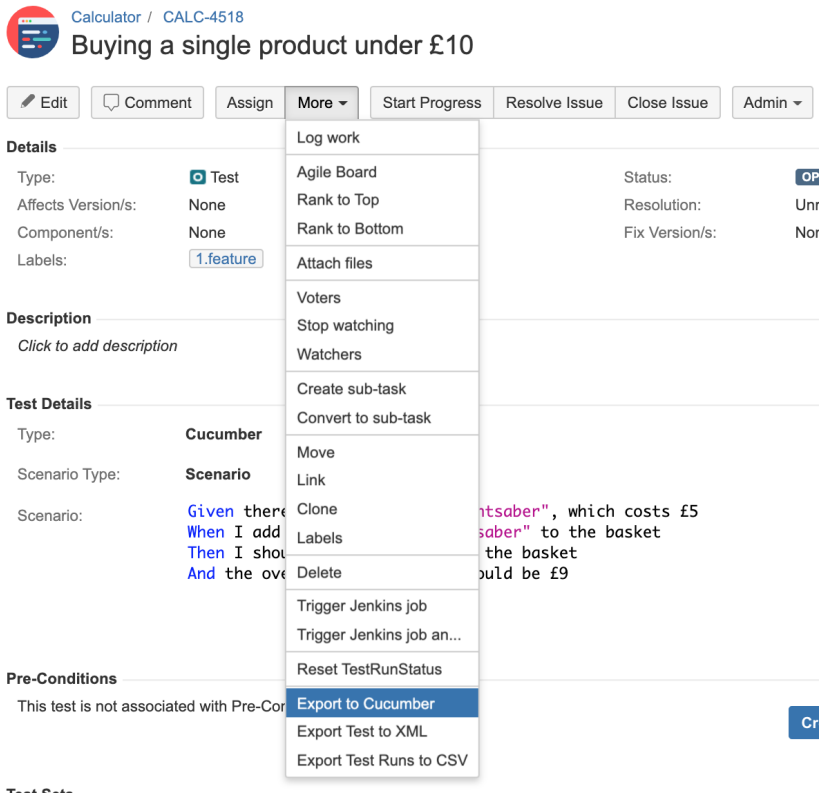
Using Jira and Xray as master

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).

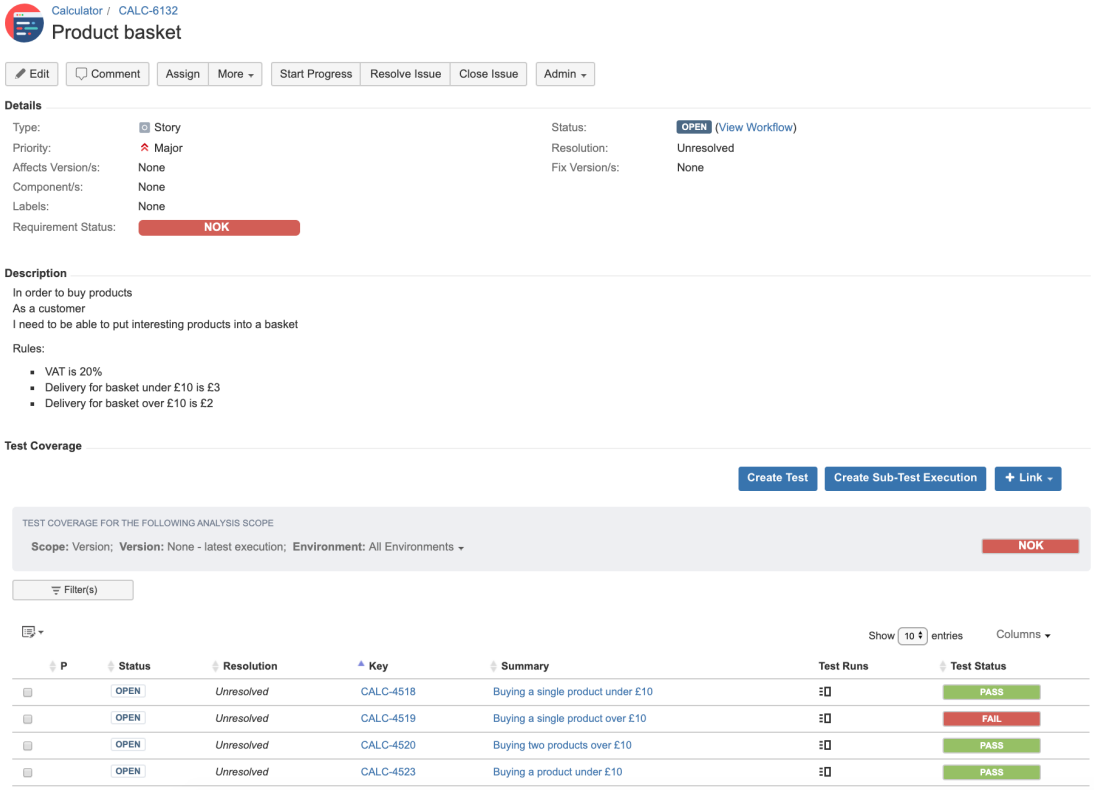
The first step is to create a "Cucumber" Test (you can also use "Behat" as a valid test type, as long as it has been defined as a possible option for the Test Type custom field).

The test is quite self-explanatory, which is the ultimate purpose of using this approach: given an existing item, add it to the basket and check the basket.

After creating the Test in Jira and associating it with requirement(s), etc., you can export the specification of the test to a ".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 contain the references to the Test issue key(s) and the covered requirement issue key, besides the scenario specification.

The following example shows a feature with 4 scenarios, which correspond to 4 Test issues in Xray.

features/1_CALC-6132.feature

```
@REQ_CALC-6132
Feature: Product basket
        #In order to buy products
        #  As a customer
        #  I need to be able to put interesting products into a basket
        #
        #  Rules:
        #  - VAT is 20%
        #  - Delivery for basket under £10 is £3
        #  - Delivery for basket over £10 is £2


        @TEST_CALC-4523 @CALC-4592 @CALC-5022 @1.feature @xpto
        Scenario Outline: Buying a product under £10
                Given there is a "<product>", which costs £<price>
                When I add the "<product>" to the basket
                Then I should have 1 product in the basket
                And the overall basket price should be £<total>

                        Examples:
                                | product | price | total |
                                | pen     | 5     | 9     |
                                | book    | 4     | 7.8   |


        @TEST_CALC-4520 @CALC-4592 @CALC-5022 @1.feature
        Scenario: Buying two products over £10
                Given there is a "Sith Lord Lightsaber", which costs £10
                And there is a "Jedi Lightsaber", which costs £5
                When I add the "Sith Lord Lightsaber" to the basket
                And I add the "Jedi Lightsaber" to the basket
                Then I should have 2 products in the basket
                And the overall basket price should be £20


        @TEST_CALC-4519 @CALC-4592 @CALC-5022 @1.feature
        Scenario: Buying a single product over £10
                Given there is a "Sith Lord Lightsaber", which costs £15
                When I add the "Sith Lord Lightsaber" to the basket
                Then I should have 1 product in the basket
                And the overall basket price should be £19


        @TEST_CALC-4518 @CALC-4592 @CALC-5022 @1.feature
        Scenario: Buying a single product under £10
                Given there is a "Sith Lord Lightsaber", which costs £5
                When I add the "Sith Lord Lightsaber" to the basket
                Then I should have 1 product in the basket
                And the overall basket price should be £9
```

The automation glue (i.e. the code corresponding to each one of these sentences - our step definitions) lives outside Jira and resides typically in some version control system, such as Git.

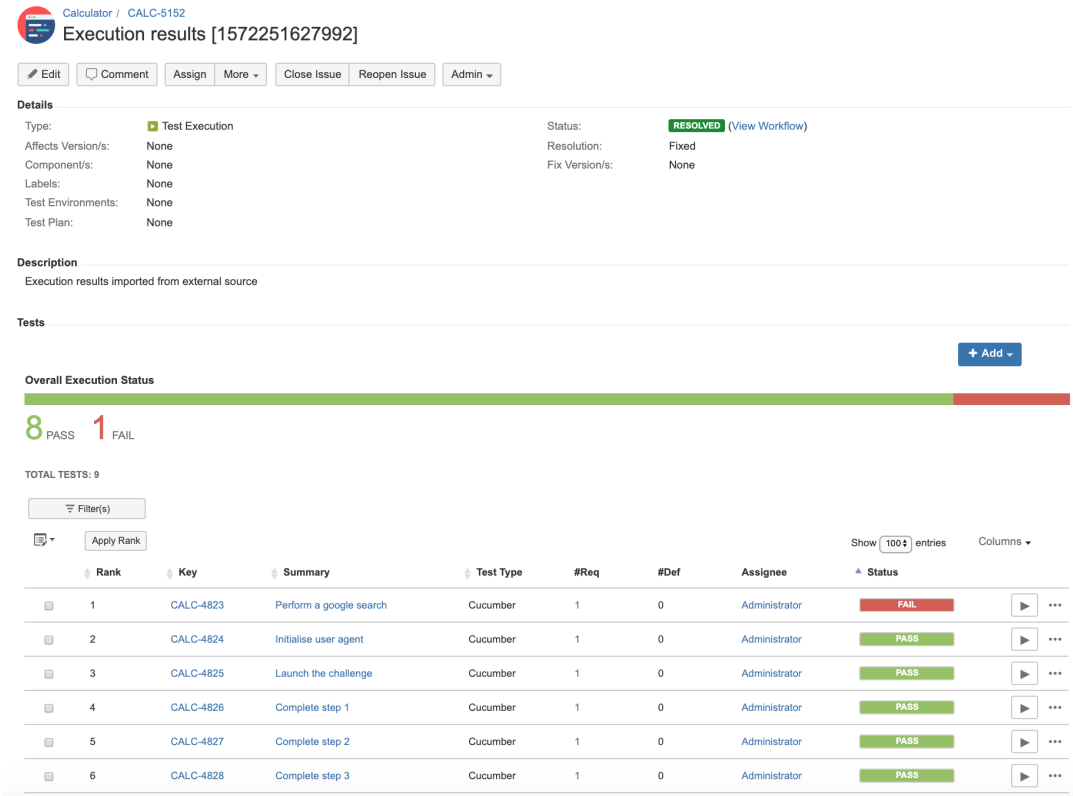
In this case, it is stored in a file name `features/bootstrap/FeatureContext.php`.

features/bootstrap/FeatureContext.php

```php
<?php

// features/bootstrap/FeatureContext.php

use Behat\Behat\Tester\Exception\PendingException;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;
use PHPUnit\Framework\Assert;

class FeatureContext implements SnippetAcceptingContext
{
    private $shelf;
    private $basket;

    public function __construct()
    {
        $this->shelf = new Shelf();
        $this->basket = new Basket($this->shelf);
    }

    /**
     * @Given there is a :product, which costs £:price
     */
    public function thereIsAWhichCostsPs($product, $price)
    {
        $this->shelf->setProductPrice($product, floatval($price));
    }

    /**
     * @When I add the :product to the basket
     */
    public function iAddTheToTheBasket($product)
    {
        $this->basket->addProduct($product);
    }

    /**
     * @Then I should have :count product(s) in the basket
     */
    public function iShouldHaveProductInTheBasket($count)
    {
        Assert::assertCount(
            intval($count),
            $this->basket
        );
    }

    /**
     * @Then the overall basket price should be £:price
     */
    public function theOverallBasketPriceShouldBePs($price)
    {
        Assert::assertSame(
            floatval($price),
            $this->basket->getTotalPrice()
        );
    }
}
```

After running the tests and generating the "Cucumber" compatible JSON report (e.g., report.json), it can be imported to Xray via the REST API, or the **Import Execution Results** action within the Test Execution, or by using one of the available CI plugins.

```
vendor/bin/behat -f cucumber_json
curl -H "Content-Type: application/json" -X POST -u admin:admin --data @"reports/report.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.

Note that the following test has a bug on purpose; in this case, the bug is not in the implementation but on the actual specification of the scenario.

Calculator / Test Execution: CALC-6131 / Test: CALC-4519

Buying a single product over £10

Import Execution Results | Export to Cucumber | Return to Test Execution | Previous | Next

Test Description

Test Issue Links (1)

tests — CALC-6132 Product basket — OPEN

Test Details

Test Type: Cucumber

Scenario Type: Scenario

Scenario:

```
Given there is a "Sith Lord Lightsaber", which costs £15
When I add the "Sith Lord Lightsaber" to the basket
Then I should have 1 product in the basket
And the overall basket price should be £19
```

Results

Context	Duration	Status
-	1.000 ms	FAIL
Steps		
Given there is a "Sith Lord Lightsaber", which costs £15	-	PASS
When I add the "Sith Lord Lightsaber" to the basket	-	PASS
Then I should have 1 product in the basket	-	PASS
And the overall basket price should be £19	1.000 ms	FAIL

```
Failed asserting that 20.0 is identical to 19.0.
```

As shown above, detailed error messages can be tracked per each step.

On the “requirement”/user story side (i.e the “feature”) we can also see how this result impacts the coverage; in this case, the story/feature is "NOK" because the latest result for the test "Buying a single product over 10" is FAIL.

Calculator / CALC-6132

Product basket

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:	NOK		

Description

In order to buy products
As a customer
I need to be able to put interesting products into a basket

Rules:

- VAT is 20%
- Delivery for basket under £10 is £3
- Delivery for basket over £10 is £2

Test Coverage

Create Test | Create Sub-Test Execution | + Link

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

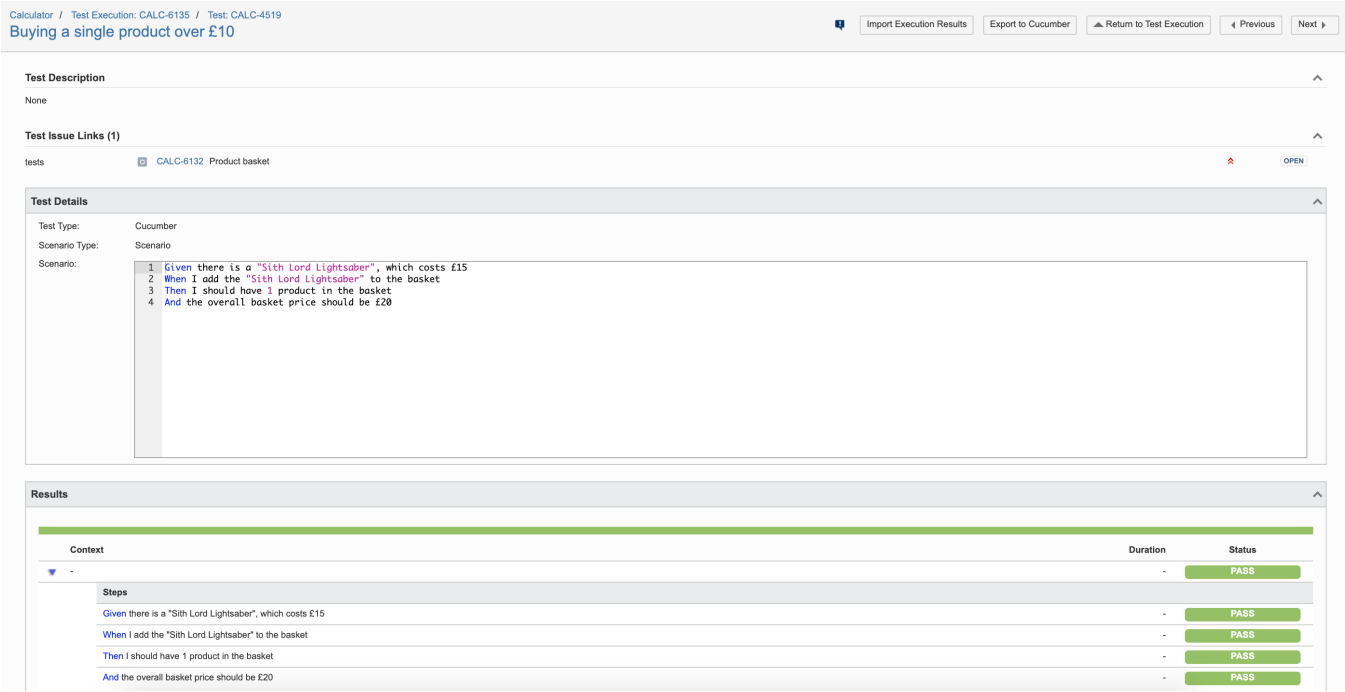
Scope: Version; **Version:** None - latest execution; **Environment:** All Environments — NOK

Filter(s)

Show 10 entries — Columns

P	Status	Resolution	Key	Summary	Test Runs	Test Status
	OPEN	*Unresolved*	CALC-4518	Buying a single product under £10		PASS
	OPEN	*Unresolved*	CALC-4519	Buying a single product over £10		FAIL
	OPEN	*Unresolved*	CALC-4520	Buying two products over £10		PASS
	OPEN	*Unresolved*	CALC-4523	Buying a product under £10		PASS

If we wanted to correct the previous error, in this case, we would need to correct the last Gherkin step of the failing scenario and run the tests again.



Using Git or other VCS as master

You can edit your .feature and .meta files outside of Jira by 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 over 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.

```
cd features
rm features.zip
zip -R features.zip -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 the correct entities in Xray (as shown in the first scenario above).

References

- https://behat.org
- http://behat.org/en/latest/quick_start.html
- https://github.com/Vanare/behat-cucumber-formatter
- 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