Acceptance & Integration Testing With Behat

The supplied code contains some working php code that fetches conference rows from the database, maps them to objects and then renders them into a html table.

Our task is to build the next feature in this amazing application: the ability to view a conference sessions. When we are finished users will be able to see a list of sessions that a conference has.

Over the next three exercises we will write Behat tests to define the acceptance criteria for our new feature. We will then write and execute acceptance tests, allow them to fail and then develop the functionality to enable them to pass. This process is known as Behaviour Driven Development or BDD.

Section One: Writing Your First Test With Behat

Exercise 1: Writing the Feature File

Write a behat test for the method 'findSessions($conferenenceId)' of the Session Service class. The method stub is already in *'lib/PHPCon/Session/Service.php'*

1. Go to the directory *'tests/acceptance/features'*
2. Open the files ' *ConferenceApi.feature'* and *'SessionApi.feature'*
3. Look at how ConferenceApi.feature is structured.
4. Create a feature in SessionApi.feature
    - Remember to include a Feature Title and Description
    - Remember to include a scenario name
    - Use Given, When & Then steps to create the scenario
    - Remember to reuse steps when possible

5. Run your feature using the behat command.

Exercise 2: Writing Steps

Running your feature file should have produced some steps. By copying these into the correct location and completing them we will have a fully functional test!

1. Copy the steps from the output into the feature context file. *'tests/acceptance/features/bootstrap/FeatureContext.php'*
2. Run the behat command again. Notice the change in output.
3. Fill in the steps.
4. Run the behat command again.
5. You should now have a failing test.

Exercise 3 – Complete the code!

You now have a failing test. This task is about getting the method to produce the correct behaviour in order for the test to pass.

Complete the body of the method under test (and the method in the . You'll know when you have it right as you will have a passing test and the functionality will work on the website.

Section Two: Writing a Behat UI test using Mink & Goutte

The last exercise used Behat to test the API (or service layer) of our application. Together with MINK Behat can test the UI of our application.

The section will test the UI of our application using the Goutte driver for Mink. Goutte is a headless browser, it sends an HTTP request and parses the raw HTML returned.

Exercise One – Using Minks Bundled Steps

1. Open the files 'tests/acceptance/features/SessionUI.feature' and 'tests/acceptance/features/ConferenceUI.feature'.
2. Look at how the first scenario scenario is implemented in ConferenceUI.feature.
3. Write a scenario to verify that sessions are output on the conference page.

4. Use Given, When, Then steps, remembering to reuse where possible.
5. You can achieve this first task using Minks built in steps.

At the end of this exercise you should have a passing test and should not have had to write any new steps.

Exercise Two – Using Custom Steps

1. Look at the way the second scenario is implemented in ConferenceUI.feature.
2. Write a scenario to verify that sessions are output on the conference page. However you now should alter the code in UIContext.php to enable the use of higher level steps built on top of Minks original steps.
   NB – This provides the benefit of abstracting away the UI implementation (css selectors) from the feature file. A major bonus!
3. Run the behat command on sessionUI.feature.

At the end of this exercise you should have two passing tests in SessionApi.feature.

Section Three: Browser Testing With Behat, Mink & Sahi

Sahi also tests the UI of our application however it does this with an actual browser. The advantage of this is that it can test javascript in addition to the presence of elements in the html.

Exercise One – Using Sahi

1. Start the sahi program:
   Windows: Use the start Menu
   Mac & Linux run the Sahi shell script (sudo sh sahi.sh)

2. Run the behat command on conferenceUI.feature. The last test is a javascript test which runs using sahi. Note the @javascript tag present at the top of the scenario. This alerts behat to the need to run the test using sahi.
3. Write a scenario to verify that there are three conferences on the page.
4. Use Given, When and Then Steps. Remembering to reuse where possible.

5. This task will involve writing a step similar to UIContext::iWaitForTheSuggestionBoxToAppear().
6. You should use the session object to execute some javascript as in the method above to verify conferences are present on the page.

NB – Normally you would test this behaviour using Goutte. This task is simply a demonstration of Sahi's capabilities.

At the end of this exercise you should have a test which opens a browser to verify that three conferences are present on the page. The test should pass.

Session Four: Dynamic Fixture Creation Using Phabric

The PhabricContext.php and PhabricExample.feature files contain working examples of Phabric's feature set.

Use any remaining time to experiment with dynamically inserting and updating data with Phabric. Write some simple scenarios which insert conferences and sessions and check they appear on the correct pages.