Writing features for Behat

The Behat features will be stored in the features/ folder in the project root. Each \*.feature file should correspond directly to a "user story" from the product backlog, and should be named with a descriptive label. For example, a homepage slideshow story from the backlog would be located in the features/ folder and named "homepage-slideshow.feature".

Just as the user stories from the backlog might be grouped into larger "epics", the Behat features can grouped likewise in subdirectories, to mimic the structure of the backlog. For example, the above test might be more specifically located at: "features/homepage/homepage-slideshow.feature". This is fluid and arbitrary - Behat will find all of the tests regardless of the folder structure.

Behat features consist of a free-form preamble, followed by any number of "scenarios". The preamble directly corresponds to the Agile concept of the "user story". The scenarios can be thought of as lists of steps for testing each "acceptance criteria" for the story.

The feature preamble

This must start with the word "Feature", followed by a one-line description of the feature. This is followed by more details, on any number of lines. In Behat it usually takes this form:

Feature**:** A one**-**line description of my feature**.** In order to **[**realize a named business value**]** As **[**an explicit system actor**]** I want to **[**gain some beneficial outcome which furthers the goal**]**

But other formats are fine, as long as value, role, and outcome are all covered. This can probably be copy/pasted from the user story.

The scenarios

Following the preamble are any number of "scenarios", which can correspond to acceptance criteria (but are not limited to those). These are the parts of the file that Behat actually parses, so the structure is a little more important. They start with the word "Scenario" followed by a one-line description of the test case. Following this are any number of "Given" steps, "When" steps, and "Then" steps. For example:

Feature**:** Serve coffee In order to earn money Customers should be able to buy coffee at all times Scenario**:** Buy last coffee Given there are 1 coffees left **in** the machine And I have deposited 1 dollar When I press the coffee button Then I should be served a coffee

Note that in the above there are essentially 2 "Given" steps, but the second one just uses "And". This is fine and makes the scenario more readable. "Given", "When", and "Then" can always be replaced with "And".

Important: The wording and language used should be at the general technical level of the type of user that the feature is intended for. In other words, if you were to print out a hardcopy of the feature and hand it to the user, they should be able to go through all the steps manually, without any confusion, and confirm the success or failure of the tests.

Pre-defined Steps

Normally, each step will require a back-end developer to code the a PHP equivalent of the step. However, there are already be many pre-defined steps that can be used out-of-the-box, which will be a time-saver for the back-end developers. Whenever it makes sense to use a pre-defined step, definitely do so. To see a full list of pre-defined steps, you can type:

bin/behat -di

But note that although using the pre-defined steps is nice, it should not be done at the cost of making the language inappropriate for the type of user. If you need a more application-specific step, don't hesitate to use it instead of pre-defined steps.

Examples of bad features, ie, what not to do

Here are some examples of bad scenarios, and why:

Scenario**:** I can always see the logo on the home page Given I am on the homepage Then I should see the "#site-logo" element

The above scenario is bad because it refers to a CSS id, "#site-logo". This kind of language means nothing to a non-developer, so if you give this feature to a user, they would not be able to manually confirm whether or not the test passes. Instead, a better step would be Then I should see the logo.

Scenario**:** I can always see the logo on every page Given I am on any page on the site Then I should see the logo

The above scenario is bad because the "Given" is too vague. If you gave this to a regular user to test, it would not be clear when that criteria ("I am on any page of the site") was satisfied. Does the user need to go to 3 pages? 5? 100? Which pages? The test would also be inconsistent because the specific pages are not indicated. A better step would specify a particular page, or use a Scenario Outline to run the test on multiple pages.

Scenario**:** The logo displays correctly on article and author pages Given I am on an article page And I should see the logo And when I click on the author's name Then I should see the logo

The above scenario is bad because it is trying to combine 2 scenarios into 1. If you find yourself adding "And when" or "And given" or similar, you should probably move that into a separate scenario.

Tagging features

You can include any number of tags at the top of a .feature file on the first line, with the @ symbol like so:

@homepage @phase1 @1234

Or whatever tags you want to use. The purpose of this is so that you can run only those features. Normally you run Behat tests with bin/behat, but to run only the homepage features, you might use bin/behat --tags 'homepage'. The "[1234](https://bitbucket.org/1234/)" above is intended to be the Clarizen item ID, which might also be a useful way to tag features.

Workflow and Getting the features to a back-end developer

Back-end developers will need the *.feature file before they can begin developing. Ideally all*.feature files for a sprint will be available to the back-end developer as soon as the sprint begins, to minimize waiting. This means that the preparation of the \*.feature files should be done towards the end of each sprint, once the next sprint's backlog is finalized.

Features should be attached to the appropriate Clarizen ticket/item so that the developer can enter it into version control and begin their work.