Drupal Testing Crash Course

DrupalCamp Colorado 2020

https://github.com/WidgetsBurritos/drupal-test-writing

David Stinemetze

- Software Developer V at Rackspace Technology
- Github/Drupal.org: @WidgetsBurritos
- Twitter: @davidstinemetze

rackspace technology.

Test D9 Site

Training Objectives

- Learn why testing is important
- Learn different types of tests (i.e. Unit/Integration/System/Acceptance)
- Learn how to write and run tests using PHPUnit, Nightwatch.js and Behat

rackspace technology.

Training Outline

What to expect from today's training

Schedule:

- **8:30AM – 10:25AM**
  - Introduction to Testing Concepts
  - Unit Testing
  - Integration Testing
- **10:25AM – 10:35AM**
  - – BREAK –
- **10:35AM – 12:30PM**
  - System Testing
  - Acceptance Testing
  - Recap

rackspace technology.

Test D9 Site

Important information

- This training will be a combination of **Lecture** and **Lab**.
- All Training materials, including these slides, are available on GitHub:
  - https://github.com/WidgetsBurritos/drupal-test-writing
  - **PowerPoint Presentation Password:** Colorado2020
- Please follow the **Getting Started** instructions on the GitHub project page, if you haven't already, to get your system prepared for today's labs.
- Individual lab assignments have been created as wiki pages on the GitHub project:
  - https://github.com/WidgetsBurritos/drupal-test-writing/wiki
- All custom code and tests live in: `web/modules/custom/my_testing_module`
- To help minimize distractions, I will be the only person on camera.
  - If you have question use the chat features within Hopin.

rackspace technology

Training Prerequisites

What you need to have for today's training

- Basic understanding of object-oriented development (preferably in the context of PHP and Drupal 8/9)
- Docker 18.06+
- Latest version of DDEV-Local
- An account on github.com *(optional)*

rackspace technology.

Introduction to Testing

Intro to Testing

What is Software Testing?

- **Software testing** is an investigation done to help stakeholders with information about the quality of the product or service under test.
- **Three main activities of testing**:
  1. **Verification** – Are we building the system right?
  2. **Validation** – Are we building the right system?
  3. **Error Detection** – Can we make things go wrong?

SOURCE: SW Testing Concepts: What is Software Testing
https://sites.google.com/site/swtestingconcepts/home/what-is-software-testing

rackspace technology

Intro to Testing

Testing as Documentation

- Software documentation is often not up-to-date.
- Documentation is often difficult to maintain.
- If tests are added and adjusted as functionality is built and modified, tests can serve as a form of documentation of how that code is supposed to function.
- For this to be true, tests should be:
  1. **Comprehensive**
  2. **Run frequently**
  3. **Consistently passing**
  4. **Easy to understand**
- This isn't always the case, and sometimes tests can be more confusing than the code itself (especially with Drupal).
- Tests can also serve as examples of how to write other tests.

SOURCE: Automated Tests as Documentation
http://swreflections.blogspot.com/2013/06/automated-tests-as-documentation.html

rackspace technology

Intro to Testing

Why don't more developers write tests?

- "I don't know how to write tests"
  - By the end of this training, you will have enough information to get started.
- "Writing tests is too hard"
  - Test writing is not without it challenges, but at the end of the day, it's just code; something you're already doing.
- "I don't have time to write tests"
  - Sure, there is more time needed to invest up front, especially when you're first learning, but the amount of time it will save you in the future by helping to minimize bugs and reinforce scalable development habits make it worth it.
- "I don't know what to test"
  - By the end of this training, hopefully you will have a few ideas on what kind of code needs to be tested, and what kinds of tests to use.
- "This code is too simple to test"
  - Sometimes this is true as "exhaustive testing is impossible", but if there's ever a doubt, err on the side of testing.

rackspace technology.

Intro to Testing

Types of Testing

- **Functional** – Testing the application against functional & business requirements.
  - **Unit Testing (or Component Testing)** *Verification*
    Individual units/components of software are tested.
  - **Integration Testing** *Verification*
    Interactions between integrated components are tested.
  - **System Testing** *Verification*
    Entire integrated system is tested.
  - **Acceptance Testing** *Validation*
    System is tested against user's acceptance criteria.

- **Non-Functional** – Testing the application against non-functional aspects of the system. Examples include performance, load and penetration testing. We won't cover non-functional testing in this training.

SOURCE: International Software Testing Qualification Board - Glossary
https://glossary.istqb.org

rackspace technology

Intro to Testing

Types of Testing in Drupal 8/9

- Drupal Core provides support for the following types of functional tests:
  - **Unit Tests** *Unit Testing*
    Base class: `\Drupal\Tests\UnitTestCase`
  - **Kernel Tests** *Integration Testing*
    Base classes:
    - `\Drupal\KernelTests\KernelTestBase`
    - `\Drupal\KernelTests\EntityKernelTestBase`
  - **Browser & JavaScript Tests** *System Testing*
    Sometimes called Functional Tests

    Base classes:
    - `\Drupal\Tests\BrowserTestBase` (No JavaScript)
    - `\Drupal\FunctionalJavascriptTests\WebDriverTestBase` (JavaScript*)

**Alternatively, JavaScript Browser testing can be performed using Nightwatch.js*

SOURCE: Types of Tests in Drupal 8
https://www.drupal.org/docs/8/testing/types-of-tests-in-drupal-8

rackspace technology

Intro to Testing

Acceptance Testing in Drupal 8/9 using behat

- **Behat** can be used for user acceptance testing in Drupal 8/9.
- Behat uses gherkin syntax, which uses natural language instead of logic to express acceptance criteria.

SOURCE: The gherkin language
http://behat.org/en/latest/user_guide/gherkin.html

SOURCE: [Meta] Use Behat for validation testing
https://www.drupal.org/project/ideas/issues/2232271

rackspace technology

Intro to Testing

How to run tests in Drupal 8/9

- You can run tests **manually**:
  - In the Drupal admin UI, via the **simpletest** module *(moved to contrib in D9)*
  - From the command line using PHP CLI, via either the **simpletest** module on an existing Drupal installation, or by using a separate **sqlite** database.
- You can also run tests **automatically**, by using a **Continuous Integration** service to trigger testing of patches and pull requests.
- Some commonly used CI services/tools:
  - **Jenkins** – Self-hosted. Integrates with GitHub, GitLab and Bitbucket via plugins. *Open Source*
  - **DrupalCI** – Runs on patches uploaded to drupal.org issues. *Uses Jenkins*
  - **TravisCI** – Third-party hosted. Integrates with GitHub.
  - **CircleCI** – Third-party hosted. Integrates with GitHub, Bitbucket.
  - **GitLab CI/CD** – Third-party hosted. Native support for GitLab.
  - **GitHub Actions** – Third-party hosted. Native support for GitHub.

SOURCE: Running tests through command-line with run-tests.sh
https://www.drupal.org/docs/8/phpunit/running-tests-through-command-line-with-run-testssh

rackspace technology

Unit Testing

Unit Testing

What is Unit Testing?

- A **unit** (or component) is the smallest part of a system that can be tested.
  - Typically, a unit corresponds to a single-purpose function.
- **Unit Testing** (or Component Testing) is the testing of individual units of software.
  - If other components are used within a unit of code, those other components are **mocked**, meaning their responses are simulated. This allows us to focus on the code we want to test, instead of external dependencies.

Type of Test	Pros	Cons
Unit	• Verify individual parts • Quickly find problems in code • Fast execution • No system setup for the test run	• Refactoring might require tests to be rewritten • Complicated mocking • No guarantee that the whole system actually works

SOURCE: International Software Testing Qualification Board - Glossary
https://glossary.istqb.org

SOURCE: Types of Tests in Drupal 8
https://www.drupal.org/docs/8/testing/types-of-tests-in-drupal-8

rackspace technology

Unit Testing

PHPUnit

Unit Testing in Drupal

- Drupal 8/9 use **PHPUnit** as its unit testing framework.
  - Drupal 7 and older used **Simpletest** for unit testing.
  - This was deprecated in Drupal 8 and moved to contrib in Drupal 9.
- Unit tests **extend** `\Drupal\Tests\UnitTestCase`
- Unit tests **live in a directory** within your module, profile or theme called: `tests/src/Unit`
- Tests will correspond to this **namespace**: `\Drupal\Tests\$extension\Unit`
- All test class names should end with `Test`.
- All test methods should begin with `test`.
  - Methods starting with anything else will not be tested.
- Test methods should make **assertions**, which define the expectations of the components under test.
- A unit test should only test one thing at a time. If you want to test a function based on a multiple inputs, multiple test methods should be added.

SOURCE: PHPUnit file structure, namespace, and required metadata
https://www.drupal.org/docs/8/phpunit/phpunit-file-structure-namespace-and-required-metadata

Unit Testing

PHPUnit

Common Assertions

- `assertTrue(bool $condition[, string $message = ''])`
- `assertFalse(bool $condition[, string $message = ''])`
- `assertEquals(mixed $expected, mixed $actual[, string $message = ''])`
- `assertSame(mixed $expected, mixed $actual[, string $message = ''])`
- `assertNull(mixed $variable[, string $message = ''])`
- `assertCount($expectedCount, $haystack[, string $message = ''])`
- `assertGreaterThan(mixed $expected, mixed $actual[, string $message = ''])`
- Most assertions have a negation method that use the same parameters. For example:
  - `assertSame()` vs `assertNotSame()`

SOURCE: PHPUnit 6.5 - Appendix A. Assertions
https://phpunit.de/manual/6.5/en/appendixes.assertions.html

rackspace technology

Unit Testing

PHPUnit

Example Unit Test

core/modules/views_ui/tests/src/Unit/Form/Ajax/RearrangeFilterTest.php:

```php
<?php

namespace Drupal\Tests\views_ui\Unit\Form\Ajax;

use Drupal\Tests\UnitTestCase;
use Drupal\views_ui\Form\Ajax\RearrangeFilter;

/**
 * Unit tests for Views UI module functions.
 *
 * @group views_ui
 */
class RearrangeFilterTest extends UnitTestCase {

  /**
   * Tests static methods.
   */
  public function testStaticMethods() {
    // Test the RearrangeFilter::arrayKeyPlus method.
    $original = [0 => 'one', 1 => 'two', 2 => 'three'];
    $expected = [1 => 'one', 2 => 'two', 3 => 'three'];
    $this->assertSame(RearrangeFilter::arrayKeyPlus($original), $expected);
  }

}
```

rackspace technology.

Unit Testing

PHPUnit

Fixture Methods

- A fixture is the known default state across all tests within a class.
- Setting up testing fixtures
  - `setUp()` – Runs prior to each individual test within a test class.
  - `tearDown()` – Runs after each individual test within a test class. Generally speaking, you only need to do this if using external resources such as files and sockets.
- Sharing fixtures across tests
  - `setUpBeforeClass()` – Runs prior to the first test run within a class.
  - `tearDownAfterClass()` – Runs after the last test run within a class.
  - Sharing fixtures across tests is generally discouraged as unit tests should be decoupled from one another. Examples where this might make sense is when using a database connection across multiple tests.

SOURCE: PHPUnit - Chapter 4. Fixtures
https://phpunit.de/manual/6.5/en/fixtures.html

rackspace technology

Unit Testing

PHPUnit

Stubs & Mocks

- **Test Double** is any pretend object used in place of a real object in tests.
  - The term is a play on “Stunt Double” from movies
- Two main types of test doubles:
  - **Stubs** override methods to provide canned responses.
    - Example: Returning the correct result of some long running method.
  - **Mocks** specify outline of full expected behavior within a method.
    - Example: Ensuring method calls other component method an explicit amount of times, and with what parameters.

SOURCE: Mocks Aren’t Stubs
https://martinfowler.com/articles/mocksArentStubs.html

Unit Testing

PHPUnit

Stub Example

```php
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testReturnSelf()
    {
        // Create a stub for the SomeClass class.
        $stub = $this->createMock(SomeClass::class);

        // Configure the stub.
        $stub->method('doSomething')
             ->will($this->returnSelf());

        // $stub->doSomething() returns $stub
        $this->assertSame($stub, $stub->doSomething());
    }
}
?>
```

SOURCE: PHPUnit 6.5 – Chapter 9. Test Doubles
https://phpunit.de/manual/6.5/en/test-doubles.html

rackspace technology

Unit Testing

PHPUnit

Mock Example

```php
<?php
use PHPUnit\Framework\TestCase;

class FooTest extends TestCase
{
    public function testFunctionCalledTwoTimesWithSpecificArguments()
    {
        $mock = $this->getMockBuilder(stdClass::class)
                     ->setMethods(['set'])
                     ->getMock();

        $mock->expects($this->exactly(2))
             ->method('set')
             ->withConsecutive(
                 [$this->equalTo('foo'), $this->greaterThan(0)],
                 [$this->equalTo('bar'), $this->greaterThan(0)]
             );

        $mock->set('foo', 21);
        $mock->set('bar', 48);
    }
}
?>
```

SOURCE: PHPUnit 6.5 – Chapter 9. Test Doubles
https://phpunit.de/manual/6.5/en/test-doubles.html

rackspace technology

Integration Testing

Integration Testing

Integration Testing in Drupal

- **Integration testing** verifies the interactions between components.
- Drupal uses **Kernel tests** for integration testing
- Kernel tests are based on PHPUnit, but are much more elaborate than unit tests.
- Any module can be enabled but installation procedures aren't run by default.
  - Module dependencies aren't automatically installed
  - Module configuration isn't installed
  - Entity types aren't created
  - Database schema isn't installed

Type of Test	Pros	Cons
Kernel	• Verify that components actually work together • Somewhat easy to locate bugs	• Slower execution • System setup required • No guarantee that end user features actually work

SOURCE: Types of Tests in Drupal 8
https://www.drupal.org/docs/8/testing/types-of-tests-in-drupal-8

rackspace technology.

Integration Testing

Kernel Tests

- Kernel tests **extend** one of these two base classes, or some derivative of them:
  - `\Drupal\KernelTests\KernelTestBase` Standard kernel test base class
  - `\Drupal\KernelTests\EntityKernelTestBase` Useful for testing entities
- Kernel tests **live in a directory** within your module, profile or theme called: `tests/src/Kernel`
- Tests will correspond to this **namespace**: `\Drupal\Tests\$extension\Kernel`
- All test class names should end with `Test`.
- All test methods should begin with `test`.
  - Methods starting with anything else will not be tested.
- Test methods should make **assertions**, which define the expectations of the components under test.
- Kernel tests can often test more than one thing at a time, but you should still use multiple test cases when testing different scenarios.

SOURCE: PHPUnit file structure, namespace, and required metadata
https://www.drupal.org/docs/8/phpunit/phpunit-file-structure-namespace-and-required-metadata

rackspace technology.

Integration Testing

Kernel Tests

Common Properties and methods

- The same fixture methods used in unit tests are available to kernel tests:
  `setUp()` / `setUpBeforeClass()` / `tearDown()` / `tearDownBeforeClass()`
- The **`$modules`** variable is used to define which modules should be installed.
  For example:
  ```
  public static $modules = ['dblog', 'system', 'user'];
  ```
- **The `$container`** variable is used to grab the service container. For example:
  ```
  $this->container->get('router.builder');
  ```
- **`installConfig()`** is used to install default configuration for the specified modules.
  For example:
  ```
  $this->installConfig(['system', 'user']);
  ```
- **`installEntitySchema()`** is used to install entity schema (i.e. entity database tables) for the specified entity type. For example:
  ```
  $this->installEntitySchema('user');
  ```
- **`installSchema()`** is used to install specified database tables from the specified module.
  For example:
  ```
  $this->installSchema('dblog', ['watchdog']);
  ```
- **`config()`** is used to interact with system config. For example:
  ```
  $this->config('system.performance')->get();
  ```

rackspace technology

SOURCE: Drupal.org - abstract class KernelTestBase
https://api.drupal.org/api/drupal/core%21tests%21Drupal%21KernelTests%21KernelTestBase.php/class/KernelTestBase/8.7.x

Integration Testing

Kernel Tests

Example

./core/modules/field/tests/src/Kernel/String/UuidItemTest.php

```php
<?php

namespace Drupal\Tests\field\Kernel\String;

use Drupal\entity_test\Entity\EntityTest;
use Drupal\Tests\field\Kernel\FieldKernelTestBase;
use Drupal\Component\Uuid\Uuid;

/**
 * Tests the UUID field.
 *
 * @group field
 */
class UuidItemTest extends FieldKernelTestBase {

  /**
   * Tests 'uuid' random values.
   */
  public function testSampleValue() {
    $entity = EntityTest::create([]);
    $entity->save();

    $uuid_field = $entity->get('uuid');

    // Test the generateSampleValue() method.
    $uuid_field->generateSampleItems();
    $this->assertTrue(Uuid::isValid($uuid_field->value));
  }

}
```

rackspace technology.

Labs 1 & 2
Writing Unit & Kernel Tests

System Testing

System Testing

System Testing in Drupal 8/9

- **System testing** tests the entire system.
- Drupal uses **Browser and JavaScript (Functional) tests** for system testing
- Browser tests are still based on PHPUnit, but are much more elaborate than unit tests, but maybe a little simpler than kernel tests.
- Tests run against installation profiles, which installs a subset of modules.
  - Drupal provides several testing profiles out-of-the-box.
  - Testing profiles are preferred for functional tests over other profiles, because functional testing is extremely slow, and they provide the lightest bootstrap option.
- Additional modules can be enabled when the test is set up.

Type of Test	Pros	Cons
Browser & JavaScript	• Verify that the system works as experienced by the user • Verify that the system works when code is refactored	• Very slow execution • Heavy system setup • Hard to locate origins of bugs • Prone to random test fails • Hard to change

SOURCE: Types of Tests in Drupal 8
https://www.drupal.org/docs/8/testing/types-of-tests-in-drupal-8

rackspace technology.

System Testing

Functional Tests

- Functional tests **extend** one of these two base classes, or some derivative class:
  - `\Drupal\Tests\BrowserTestBase` ***No Javascript***
  - `\Drupal\FunctionalJavascriptTests\WebDriverTestBase` **Javascript*

  *Drupal now supports Nightwatch.js, which is JavaScript-based. It is recommended to use that for JavaScript testing instead of WebDriverTestBase.
- Functional tests **live in a directory** within your module, profile or theme called: `tests/src/Functional(Javascript)*` or `tests/src/Nightwatch`
- Tests will correspond to this **namespace** for PHP-based tests: `\Drupal\Tests\$extension\Functional(Javascript)*`
- All test class names should end with `Test` for PHP-based tests.
- All test methods should begin with `test` For PHP-based tests.
- Test methods should make **assertions**, which define the expectations of the system functionality under test.
- System test artifacts are available in `sites/simpletest/browser_output` after running tests, which contain browser markup from your test.

SOURCE: PHPUnit file structure, namespace, and required metadata
https://www.drupal.org/docs/8/phpunit/phpunit-file-structure-namespace-and-required-metadata

rackspace technology

System Testing

Functional Tests

Common Properties and Methods

- The same fixture methods used in unit/kernel tests are available to functional tests:
  `setUp()` / `setUpBeforeClass()` / `tearDown()` / `tearDownBeforeClass()`
- The **`$profile`** variable is used to define which installation profile should be used
  `public static $profile = 'testing';`
- The **`$defaultTheme`** variable is used to define which theme should be used
  `public static $defaultTheme = 'stable';`
- The **`$modules`** variable defines which additional modules should be installed. For example:
  `public static $modules = ['dblog', 'system', 'user'];`
- **The `$container`** variable is used to grab the service container. For example:
  `$this->container->get('router.builder');`
- **`config()`** interact with system config. For example:
  `$this->config('system.performance')->get();`
- **`drupalGet()`** performs a GET request on a system route. For example:
  `$this->drupalGet('admin/config');`
- **`drupalPostForm()`** performs a POST request using a form on the route. For example:
  `$this->drupalPostForm('admin/config/people/ban', $form_values, 'Add');`
- **`drupalLogin()`** logs in the specified user. For example:
  `$this->drupalLogin($this->adminUser);`
- **`drupalCreateUser()`** creates a user. For example:
  `$this->drupalCreateUser(['administer blocks', 'administer themes']);`
- **`assertSession()`** retrieves a WebAssert object. For example:
  `$session = $this->assertSession();`

SOURCE: Drupal.org - abstract class BrowserTestBase
https://api.drupal.org/api/drupal/core%21tests%21Drupal%21Tests%21BrowserTestBase.php/class/BrowserTestBase/8.7.x

rackspace technology

System Testing

Functional Tests

Example

```php
<?php

namespace Drupal\Tests\my_testing_module\Functional;

use Drupal\Tests\BrowserTestBase;

/**
 * Functional tests for my_testing_module.
 *
 * @group my_testing_module
 */
class MyFunctionalTest extends BrowserTestBase {

  /**
   * {@inheritdoc}
   */
  public $profile = 'testing';

  /**
   * {@inheritdoc}
   */
  public $defaultTheme = 'stable';

  /**
   * Logged in user.
   *
   * @var \Drupal\Core\Session\AccountInterface
   */
  protected $authorizedUser;

  /**
   * {@inheritdoc}
   */
  public static $modules = [
    'my_testing_module',
  ];

  /**
   * {@inheritdoc}
   */
  protected function setUp() {
    parent::setUp();
    $this->authorizedUser = $this->drupalCreateUser([], 'Regular User');
  }

  /**
   * Functional test confirming the controller is loading.
   */
  public function testMessageControllerIsLoadingForAuthenticatedUsers() {
    $assert = $this->assertSession();
    $this->drupalLogin($this->authorizedUser);
    $this->drupalGet('my-message');
    $assert->pageTextContains('Hi Regular User.');
  }

}
```

rackspace technology.

System Testing

Functional Tests

Common WebAssert Methods

- **`addressEquals()`** checks that current session address is equals to provided one. For example:
  `$assert->addressEquals('admin/content/media')`
- **`buttonExists()`** checks that specific button exists on the current page. For example:
  `$assert->buttonExists('Continue');`
- **`checkboxChecked()`** checks that specific checkbox is checked. For example:
  `$assert->checkboxChecked('entity_types[menu_link_content]');`
- **`elementTextContains()`** checks that matching element (css/xpath) contains text. For example:
  `$assert->elementTextContains('css', 'div.node__submitted', 'Submitted by');`
- **`fieldExists()`** checks that specific field exists on the current page. For example:
  `$assert->fieldExists('subject[0][value]');`
- **`pageTextContains()`** checks that current page contains text. For example:
  `$assert->pageTextContains('Lorem ipsum');`
- **`responseContains()`** checks that page HTML (response content) contains text. For example:
  `$assert->responseContains('<h2>Topics</h2>');`
- **`responseMatches()`** checks that page HTML (response content) matches regex. For example:
  `$assert->responseMatches('/\<a.*title\=\"' . t('sort by Username') . '\".*\>/');`
- **`statusCodeEquals()`** checks the status code of the response. For example:
  `$assert->statusCodeEquals(403);`

rackspace technology

SOURCE: Drupal.org – class WebAssert
https://api.drupal.org/api/drupal/core%21tests%21Drupal%21Tests%21WebAssert.php/class/WebAssert/8.7.x

System Testing

Nightwatch.js

- **Nightwatch.js** is a JavaScript end-to-end testing framework.
- It is useful for testing Javascript-specific functionality in Drupal 8 & 9.
- Requires Node.js
- Requires a WebDriver service to interact with browsers:
  - GeckoDriver (Firefox)
  - ChromeDriver (Chrome)
  - Microsoft WebDriver (Edge)
  - SafariDriver (Safari)
- It is still relatively new to Drupal, so there aren't a ton of examples, but you can find a few that we've written for the *performance budget* module, which test our integration with chart.js:
  - https://git.drupalcode.org/project/performance_budget/-/tree/2.x/tests/src/Nightwatch/Tests

SOURCE: Nightwatch.js – Getting Started
https://nightwatchjs.org/gettingstarted/installation/

SOURCE: Drupal.org - JavaScript testing using Nightwatch
https://www.drupal.org/docs/8/testing/javascript-testing-using-nightwatch

rackspace technology

System Testing

Nightwatch.js Testing

Special variables and properties

rackspace technology

- Nightwatch.js test files are defined as nodejs modules thus all code belongs inside a `module.exports = {};` declaration.
- Each unique test case is defined as a property in `module.exports` using a string as the key and passing the `browser` variable into an anonymous function.
- Special Properties
  - `'@tags'` – An array of applicable tags for a set of tests, which allow you to selectively run tests
  - `before` – Runs before execution of entire test suite
  - `after` – Runs after execution of entire test suite
  - `beforeEach` – Runs before execution of individual test cases
  - `afterEach` – Runs after execution of individual test cases
- The `browser` variable is used to execute commands and perform assertions.
  - `browser.assert` – Performs assertions, ending on failure.
  - `browser.verify` – Performs assertions, continuing on failure.

SOURCE: Nightwatch.js - Using before[Each] and after[Each] hooks
https://nightwatchjs.org/guide#using-before-each-and-after-each-hooks

System Testing

Nightwatch.js Testing

Common Assertions

- **.assert.attributeContains()** – checks if an element attribute contains text
  ```
  browser.assert.attributeContains('#someElement', 'href', 'google.com')
  ```
  See also .attributeEquals()
- **.assert.containsText()** – checks if an element contains text
  ```
  browser.assert.containsText('#main', 'The Night Watch');
  ```
- **.assert.cssClassPresent()** – checks if css class exists on an element
  ```
  browser.assert.cssClassPresent('#main', 'container');
  ```
- **.assert.cssProperty()** – checks if css property has expected value
  ```
  browser.assert.cssProperty('#main', 'display', 'block');
  ```
- **.assert.elementPresent()** – checks if element is present in the DOM
  ```
  browser.assert.elementPresent('#main');
  ```
- **.assert.hidden()** – checks if element is not visible on the page
  ```
  browser.assert.hidden('.should_not_be_visible');
  ```
  See also .visible()
- **.assert.urlContains()** – checks if URL contains string
  ```
  browser.assert.urlContains('nightwatchjs.org');
  ```
  See also .urlEquals()
- **.assert.value()** – checks if form value equals string
  ```
  browser.assert.value('form.login input[type=text]', 'username');
  ```
  See also .valueContains()
- **.assert.not.*()** – negates another assertion
  ```
  browser.assert.not.cssProperty('#main', 'display', 'block');
  ```

SOURCE: Nightwatch.js – API
https://nightwatchjs.org/api/

rackspace technology

System Testing

Nightwatch.js Testing

Drupal Functions

rackspace technology

- **.drupalInstall()** – Installs test Drupal site based on specified criteria
```
.drupalInstall({setupFile: __dirname +
'/fixtures/TestSiteInstallTestScript.php'})
```
- **.drupalUninstall()** – Uninstalls test Drupal site
- **.drupalRelativeURL()** – Navigates to a URL relative to the Drupal root
```
.drupalRelativeURL('/admin/reports')
```
- **.drupalCreateRole()** – Attempts to create a new role
```
.drupalCreateRole({ permissions: ['access site reports'], })
```
- **.drupalCreateUser()** – Attempts to create a new user
```
.drupalCreateUser({ name: 'user', password: '123', permissions:
['access site reports'], })
```
- **.drupalLogin()** – Attempts to login as user
```
.drupalLogin({ name: 'user', password: '123' })
```
- **.drupalLoginAsAdmin()** – Attempts to login as admin user
- **.drupalLogout()** – Logs a user out
- **.drupalUserIsLoggedIn()** – Indicates if a user is currently logged in

SOURCE: Drupal.org - JavaScript testing using Nightwatch
https://www.drupal.org/docs/8/testing/javascript-testing-using-nightwatch

SOURCE: Drupal.org - New nightwatch commands for login and logout
https://www.drupal.org/node/2986276

System Testing

Nightwatch.js Testing

Example

./core/tests/Drupal/Nightwatch/Tests/statesTest.js:

```
module.exports = {
  '@tags': ['core'],
  before(browser) {
    browser.drupalInstall().drupalLoginAsAdmin(() => {
      browser
        .drupalRelativeURL('/admin/modules')
        .setValue('input[type="search"]', 'FormAPI')
        .waitForElementVisible('input[name="modules[form_test][enable]"]', 1000)
        .click('input[name="modules[form_test][enable]"]')
        .click('input[type="submit"]') // Submit module form.
        .click('input[type="submit"]'); // Confirm installation of dependencies.
    });
  },
  after(browser) {
    browser.drupalUninstall();
  },
  'Test form with state API': browser => {
    browser
      .drupalRelativeURL('/form-test/javascript-states-form')
      .waitForElementVisible('body', 1000)
      .waitForElementNotVisible('input[name="textfield"]', 1000);
  },
};
```

rackspace technology.

Acceptance Testing

Acceptance Testing

What is Acceptance Testing?

- **Acceptance Criteria** is what must be satisfied in order for a system or component to be accepted by users and stakeholders.
- **Acceptance Testing** is a form of testing that verifies a system or component meets acceptance criteria.
- **Types of Acceptance Testing**
  - **Contractual Acceptance Testing** – Validates system meets contractual requirements
  - **Operational Acceptance Testing** – Validates system is resilient, recoverable, manageable and maintains data integrity
  - **Regulatory Acceptance Testing** – Validates system adheres to laws, policies and regulations.
  - **User Acceptance Testing** – Validates users needs and requirements are met

SOURCE: International Software Testing Qualification Board - Glossary
https://glossary.istqb.org

SOURCE: Guru99 – What is Operational Acceptance Testing?
https://www.guru99.com/operational-testing.html

rackspace technology

Acceptance Testing

Acceptance Testing
in Drupal 8/9 with Behat

- **Behat** can be used for user acceptance testing in Drupal 8 and 9.
- Behat uses gherkin syntax, which uses natural language instead of logic to express acceptance criteria.
  - This helps non-programmers express or understand how systems work.
  - It's often easier to understand than code.
  - Can help serve as documentation for your project where more complicated testing mechanisms can't.

SOURCE: The gherkin language
http://behat.org/en/latest/user_guide/gherkin.html

SOURCE: [Meta] Use Behat for validation testing
https://www.drupal.org/project/ideas/issues/2232271

rackspace technology

Acceptance Testing

Gherkin Keywords

- `Feature` – Provides high level description of a software feature
- `Scenario` (or `Example`) – A set of steps demonstrating a business rule
- Steps
  - `Given` – Describe system context (i.e. What happened in the past)
  - `When` – Describe an event or action (i.e. What is happening now)
  - `Then` – Describe expected outcome (i.e. What will happen in the future)
  - `And`, `But` – When multiple given/when/then steps occur in a row you can combine them to improve readability
- `Background` – Share system context (i.e. Given steps) across multiple scenarios
- `Scenario Outline` (or `Scenario Template`) – Run the same scenarios multiple times with different variables

rackspace technology

SOURCE: Gherkin Reference
https://cucumber.io/docs/gherkin/reference

Acceptance Testing

Example Behat Acceptance Test using Scenario Outline

```
@data @javascript
@uat
Feature: Forms Fields.

Background:
    Given I visit "/test-page"

Scenario Outline: Google Analytics form fields are present and populated as expected.
    When I click the "#node-6686 a.cta-button" element
    Then I should see a "<theGivenElement>" element
    And the "<theGivenElement>" element should <presence> empty

Scenarios:
    | theGivenElement               | presence  |
    | input[name='gaclientid__c']   | not be    |
    | input[name='gauserid__c']     | be        |
    | input[name='gatrackid__c']    | not be    |
    | input[name='first_name']      | be        |
```

rackspace technology.

Acceptance Testing

Example Behat Acceptance Test using Custom Context

features/drupal/cache.feature:

```
Feature: Cache

Scenario:
  When I visit "/"
  Then the response should contain "Welcome to Drush Site-Install"
  And the cache tag "|http_response|" is present
  And the cache context "|url.path.is_front|" is present
```

features/bootstrap/FeatureContext.php:

```
/**
 * Asserts that the specified cache tag is present.
 *
 * @Then the cache tag :tag is present
 */
public function theCacheTagIsPresent($tag) {
  $this->assertSession()->responseHeaderMatches('X-Drupal-Cache-Tags', $tag);
}

/**
 * Asserts that the specified cache context is present.
 *
 * @Then the cache context :context is present
 */
public function theCacheContextIsPresent($context) {
  $this->assertSession()->responseHeaderMatches('X-Drupal-Cache-Contexts', $context);
}
```

rackspace technology.

Acceptance Testing

Installing Behat

- Add the following packages via composer:
  - `behat/behat`
  - `dmore/behat-chrome-extension`
  - `drupal/drupal-extension`
- Behat settings are defined in a `behat.yml` file usually at the root of your project
  - Tests live in feature files defined by a `path` directive in your behat settings
    - By default these files will leave in a `features` directory in the same folder as your `behat.yml` file.
  - Step definitions can be defined via contexts specified in the `contexts` directive in your behat settings.

rackspace technology.

Acceptance Testing

Drupal Contexts

- `drupal/drupal-extension` provides the following contexts:
  - `RawDrupalContext` – No step-definitions provided but has all the necessary pieces to interact with Drupal and the browser
  - `DrupalContext` – Provides step-definitions for creating users, terms and nodes
  - `MinkContext` – Provides steps-definitions specific to regions and forms plus basic browser simulation
  - `MarkupContext` – Provides step-definitions related to HTML tags/attributes
  - `MessageContext` – Provides step-definitions related to Drupal messages (i.e. notices, warnings, errors)
  - `DrushContext` – Allows steps to call drush commands
- You can also define your own contexts to define your own step definitions:
  - These should live in a `bootstrap` folder inside your main features directory

rackspace technology.

Acceptance Testing

Drupal Extension Drivers

- `drupal/drupal-extension` provides three different extension drivers:

Feature	Blackbox	Drush	Drupal API
Map Regions	Yes	Yes	Yes
Create users	No	Yes	Yes
Create nodes	No	No	Yes
Create vocabularies	No	No	Yes
Create taxonomy terms	No	No	Yes
Run tests and site on different servers	Yes	Yes	No

- We will mostly focus on the Drupal API driver for this training.
- To enable the Drupal API driver set `api_driver` to `drupal` and define `drupal_root` in your behat settings.
- Tag tests with `@api` to enable the Drupal API driver for that particular test.

SOURCE: Drupal Extension Drivers
https://behat-drupal-extension.readthedocs.io/en/3.1/drivers.html

rackspace technology

Labs 3 & 4
Writing System & Acceptance Tests

Common Pitfalls

Mistakes to look out for

- If PHPUnit-based tests aren't running as expected, confirm the following:
  - Test file name ends with `Test.php` and class name matches file name
  - Test methods begins with `test`
  - Ensure namespace is correct in the test for the respective test type
  - Ensure file and folder path is correct for the respective test type
- In Functional tests don't use `$this->loggedInUser` for a mock user. This property is used by `BrowserTestBase` to determine which user is actively logged in. Setting this value can cause a lot of confusion. Instead use variables like `$this->authorizedUser` and always use `$this->drupalLogin()` to log a user in.
- When creating mock users, if explicitly setting a UID, make sure to use 2 or greater, as UID 1 has permission to do everything in Drupal.
- Kernel and Browser tests require valid configuration schemas if an installed module provides default configuration settings. This is especially fun when writing tests related to other contrib modules that fail to provide a schema.
- If you experience different behavior in Kernel or Browser tests than you do in your browser, most likely you're missing a one or more modules in `$modules`.

In Review

What we covered

- What is Software Testing?
  - Verification
  - Validation
  - Error Detection
- Types of Testing
  - Functional vs Non-functional Testing
  - Unit vs Integration vs System vs Acceptance Testing
- Unit Testing with PHPUnit
  - Assertions/Fixtures
  - Stubs vs Mocks
- Integration Testing using Kernel Tests
- System Testing using Browser (or Functional) Tests
- System Testing using Nightwatch.js
- Acceptance Testing using Behat

rackspace technology.

Sources

List of resources used in this training (in order of first appearance)

- SW Testing Concepts: What is Software Testing
  https://sites.google.com/site/swtestingconcepts/home/what-is-software-testing
- Automated Tests as Documentation
  http://swreflections.blogspot.com/2013/06/automated-tests-as-documentation.html
- International Software Testing Qualification Board Glossary
  https://glossary.istqb.org
- Types of Tests in Drupal 8
  https://www.drupal.org/docs/8/testing/types-of-tests-in-drupal-8
- JavaScript testing using Nightwatch
  https://www.drupal.org/docs/8/testing/javascript-testing-using-nightwatch
- The gherkin language
  http://behat.org/en/latest/user_guide/gherkin.html
- [Meta] Use Behat for validation testing
  https://www.drupal.org/project/ideas/issues/2232271
- Running tests through command-line with run-tests.sh
  https://www.drupal.org/docs/8/phpunit/running-tests-through-command-line-with-run-testssh

rackspace technology

Sources (Continued)

List of resources used in this training (in order of first appearance)

- PHPUnit 6.5 - Chapter 4. Fixtures
  https://phpunit.de/manual/6.5/en/fixtures.html
- PHPUnit 6.5 – Chapter 9. Test Doubles
  https://phpunit.de/manual/6.5/en/test-doubles.html
- PHPUnit 6.5 - Appendix A. Assertions
  https://phpunit.de/manual/6.5/en/appendixes.assertions.html
- Martin Fowler - Mocks Aren't Stubs
  https://martinfowler.com/articles/mocksArentStubs.html
- Drupal.org - abstract class BrowserTestBase
  https://api.drupal.org/api/drupal/core%21tests%21Drupal%21Tests%21BrowserTestBase.php/class/BrowserTestBase/8.7.x
- Drupal.org – class WebAssert
  https://api.drupal.org/api/drupal/core%21tests%21Drupal%21Tests%21WebAssert.php/class/WebAssert/8.7.x
- Nightwatch.js – Getting Started
  https://nightwatchjs.org/gettingstarted/installation/
- Drupal.org - JavaScript testing using Nightwatch
  https://www.drupal.org/docs/8/testing/javascript-testing-using-nightwatch

rackspace technology

Sources (continued)

List of resources used in this training (in order of first appearance)

- Nightwatch.js - Using before[Each] and after[Each] hooks
  https://nightwatchjs.org/guide#using-before-each-and-after-each-hooks
- Drupal.org - New nightwatch commands for login and logout
  https://www.drupal.org/node/2986276
- Guru99 – What is Operational Acceptance Testing?
  https://www.guru99.com/operational-testing.html
- Gherkin Reference
  https://cucumber.io/docs/gherkin/reference
- Drupal Extension Drivers
  https://behat-drupal-extension.readthedocs.io/en/3.1/drivers.html

rackspace technology.

Drupal Testing Crash Course

DrupalCamp Colorado 2020

https://github.com/WidgetsBurritos/drupal-test-writing

David Stinemetze

- Software Developer V at Rackspace Technology
- Github/Drupal.org: @WidgetsBurritos
- Twitter: @davidstinemetze

rackspace technology.