Testing Your Zend Framework MVC Application

Matthew Weier O'Phinney
Project Lead
Zend Framework
What we'll cover

- Unit Testing basics
- Basic functional testing in ZF
- Several “Advanced” ZF testing topics
Why test?
Simplify maintenance

- Testing defines expectations
- Testing describes behaviors identified by the application
- Testing tells us when new changes break existing code and behaviors
Quantify code quality

- Code coverage exercised by unit tests
- Test methods document behaviors code should define
Psychological benefits

Warm fuzzy feeling from seeing green
Testing is not... reloading
Testing is not... var_dump()
Testing is... reproducible
Testing is... automatable
Good testing includes...

- Defined behaviors
- Code examples of use cases
- Expectations
PHP testing frameworks

- PHPT
  - Used by PHP, and some PEAR and independent libraries

- SimpleTest
  - JUnit-style testing framework

- PHPUnit
  - JUnit-style testing framework
  - De facto industry standard
Testing Basics
Writing unit tests

- Create a test class
- Create one or more methods describing behaviors
  - State the behaviors in natural language
- Write code that creates the behavior(s)
  - Write code exercising the API
- Write assertions indicating expectations
Create a test class

- Usually, named after the Unit Under Test

```php
class EntryTest extends PHPUnit_Framework_TestCase {
}
```
Write a method describing a behavior

- Prefix with “test”

class EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function testMaySetTimestampWithString()
    {
    }
}
Write code creating the behavior

class EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function testMaySetTimestampWithString()
    {
        $string = 'Fri, 7 May 2010 09:26:03 -0700';
        $ts = strtotime($string);
        $this->entry->setTimestamp($string);
        $setValue = $this->entry->getTimestamp();
    }
}
class EntryTest
    extends PHPUnit_Framework_TestCase
{

    public function testMaySetTimestampWithString()
    {
        $string = 'Fri, 7 May 2010 09:26:03 -0700';
        $ts = strtotime($string);
        $this->entry->setTimestamp($string);
        $setValue = $this->entry->getTimestamp();
        $this->assertSame($ts, $setValue);
    }
}
Run the tests

• Failure?
  ▶ Check your tests and assertions for potential typos or usage errors
  ▶ Check the Unit Under Test for errors
  ▶ Make corrections and re-run the tests

• Success?
  ▶ Move on to the next behavior or feature!
Some testing terminology
Test scaffolding

- Make sure your environment is free of assumptions
- Initialize any dependencies necessary prior to testing
- Usually done in the “setUp()” method
Test doubles

• **Stubs**
  Replacing an object with another so that the system under test can continue down a path.

• **Mock Objects**
  Replacing an object with another and defining expectations for it.
Additional testing types

- **Conditional testing**
  *Testing only when certain environmental conditions are met.*

- **Functional and Integration tests**
  *Testing that the systems as a whole behaves as expected; testing that the units interact as expected.*
Quasi-Functional testing in Zend Framework
Overview

- Setup the phpunit environment
- Create a TestCase based on ControllerTestCase
- Bootstrap the application
- Create and dispatch a request
- Perform assertions on the response
The PHPUnit environment

- Directory structure

```
tests
|-- application
| `-- controllers
|-- Bootstrap.php
|-- library
| `-- Custom
|-- phpunit.xml
```

4 directories, 2 files
The PHPUnit environment

- `phpunit.xml`

```xml
<phpunit bootstrap="/Bootstrap.php">
  <testsuite name="Test Suite">
    <directory>./</directory>
  </testsuite>
  <filter>
    <whitelist>
      <directory suffix="/php">../library/</directory>
      <directory suffix="/php">../application/</directory>
      <exclude>
        <directory suffix="/phtml">../application/</directory>
      </exclude>
    </whitelist>
  </filter>
</phpunit>
```
The PHPUnit environment

- **Bootstrap.php**

```php
$rootPath = realpath(dirname($__DIR__));
if (!defined('APPLICATION_PATH')) {
    define('APPLICATION_PATH', $rootPath . '/application');
}
if (!defined('APPLICATION_ENV')) {
    define('APPLICATION_ENV', 'testing');
}
set_include_path(implode(PATH_SEPARATOR, array('.', $rootPath . '/library', get_include_path(), )));
require_once 'Zend/Loader/Autoloader.php';
$loader = Zend_Loader_Autoloader::getInstance();
$loader->registerNamespace('Custom_');
```
Create a test case class

- Extend Zend_Test_PHPUnit_ControllerTestCase

```php
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
}
```
Bootstrap the application

- Create a Zend_Application instance, and reference it in your setUp()

```
class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    public function setUp()
    {
        $this->bootstrap = new Zend_Application(
            APPLICATION_ENV,
            APPLICATION_PATH . '/configs/application.ini');
        parent::setUp();
    }
}
```
Create and dispatch a request

• Simple method: dispatch a “url”

```php
class ExampleControllerTest extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testStaticPageHasGoodStructure()
    {
        $this->dispatch('/example/page');
        // ...
    }
}
```
Create and dispatch a request

- More advanced: customize the request object prior to dispatching

```php
class ExampleControllerTest extends Zend_Test_PHPUnit_ControllerTestCase {
    // ...
    public function testXhrRequestReturnsJson() {
        $this->getRequest()->setHeader('X-Requested-With', 'XMLHttpRequest')->setQuery('format', 'json');
        $this->dispatch('/example/xhr-endpoint');
        // ...
    }
}
```
Create assertions

• Typical assertions are for:
  ▶ Structure of response markup *Using either CSS selectors or XPath assertions.*
  ▶ HTTP response headers and status code
  ▶ Request and/or Response object artifacts
CSS Selector assertions

- `assertQuery($path, $message = '')`
- `assertQueryContentContains($path, $match, $message = '')`
- `assertQueryContentRegex($path, $pattern, $message = '')`
- `assertQueryCount($path, $count, $message = '')`
- `assertQueryCountMin($path, $count, $message = '')`
- `assertQueryCountMax($path, $count, $message = '')`

Each has a "Not" variant
XPath Selector assertions

- `assertXpath($path, $message = '')`
- `assertXpathContentContains($path, $match, $message = '')`
- `assertXpathContentRegex($path, $pattern, $message = '')`
- `assertXpathCount($path, $count, $message = '')`
- `assertXpathCountMin($path, $count, $message = '')`
- `assertXpathCountMax($path, $count, $message = '')`
- each has a "Not" variant
Redirect assertions

- `assertRedirect($message = '')`
- `assertRedirectTo($url, $message = '')`
- `assertRedirectRegex($pattern, $message = '')`
- each has a "Not" variant
Response assertions

- assertResponseCode($code, $message = "")
- assertHeader($header, $message = "")
- assertHeaderContains($header, $match, $message = "")
- assertHeaderRegex($header, $pattern, $message = "")
- each has a "Not" variant
Request assertions

- `assertModule($module, $message = '')`
- `assertController($controller, $message = '')`
- `assertAction($action, $message = '')`
- `assertRoute($route, $message = '')`
- each has a "Not" variant
Assertion examples

```php
public function testSomeStaticPageHasGoodStructure() {
    $this->dispatch('/example/page');
    $this->assertResponseCode(200);
    $this->assertQuery('div#content p');
    $this->assertQueryCount('div#sidebar ul li', 3);
}
```
public function testXhrRequestReturnsJson() {
  // ...
  $this->assertNotRedirect();
  $this->assertHeaderContains(
    'Content-Type', 'application/json');
}
Advanced Topics
or: real world use cases
Testing models and resources

- **The Problem:**
  *These classes cannot be autoloaded with the standard autoloader.*

- **The Solution:**
  *Use Zend_Application to bootstrap the “modules” resource during setUp()*
Testing models and resources

class Blog_Model_EntryTest
    extends PHPUnit_Framework_TestCase
{
    public function setUp()
    {
        $this->bootstrap = new Zend_Application(
            APPLICATION_ENV,
            APPLICATION_PATH . '/configs/application.ini');
        $this->bootstrap->bootstrap('modules');
        $this->model = new Blog_Model_Entry();
    }
}
Testing actions requiring authentication

● The Problem:
  *Some actions may require an authenticated user; how can you emulate this?*

● The Solution:
  *Manually authenticate against Zend_Auth prior to calling dispatch().*
Authenticating a test user

class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function loginUser($user)
    {
        $params = array('user' => $user);
        $adapter = new Custom_Auth_TestAdapter($params);
        $auth = Zend_Auth::getInstance();
        $auth->authenticate($adapter);
        $this->assertTrue($auth->hasIdentity());
    }
}
Authenticating and dispatching

class ExampleControllerTest
    extends Zend_Test_PHPUnit_ControllerTestCase
{
    // ...
    public function testAdminUserCanAccessAdmin()
    {
        $this->loginUser('admin');
        $this->dispatch('/example/admin');
        $this->assertQuery('div#content.admin');
    }
}
Testing pages dependent on prior actions

- The Problem:
  Some actions are dependent on others; e.g., retrieving a page with content highlighted based on a search string.

- The Solution:
  Dispatch twice, resetting the request and response between calls.
Testing pages dependent on prior actions

```php
class ExampleControllerTest extends Zend_Test_PHPUnit_ControllerTestCase {
    // ...

    public function testHighlightedTextAfterSearch() {
        $this->getRequest()->setQuery('search', 'foobar');
        $this->dispatch('/search');

        $this->resetRequest();
        $this->resetResponse();

        $this->dispatch('/example/page');
        $this->assertQueryContains('span.highlight', 'foobar');
    }
}
```
Conclusions
Test Always!

- Unit test your models, service layers, etc.
- Do functional/acceptance testing to test workflows, page structure, etc.
Zend Framework Training & Certification

- Zend Framework: Fundamentals
  This course combines teaching ZF with the introduction of the Model-View-Controller (MVC) design pattern, to ensure you learn current best practices in PHP development.

Next Class: August 16, 17, 18, 19 20, 23, 24, 25 & 26 from 9am-11am Pacific
Zend Framework Training & Certification

- Zend Framework: Advanced

The Zend Framework: Advanced course is designed to teach PHP developers already working with Zend Framework how to apply best practices when building and configuring applications for scalability, interactivity, and high performance.

Next Class: Sept. 7, 8, 9, 10, 13, 14, 15, 16 & 17 from 8:30am-10:30am Pacific
Test Prep: Zend Framework Certification

The Test Prep: Zend Framework Certification course prepares experienced developers who design and build PHP applications using ZF for the challenge of passing the certification exam and achieving the status of Zend Certified Engineer in Zend Framework (ZCE-ZF).

Free Study Guide
ZendCon 2010!

- 1–4 November 2010
- http://www.zendcon.com/