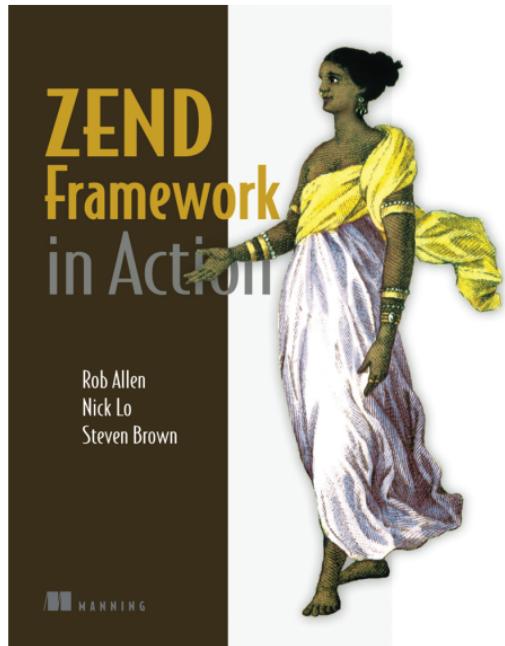


Introducing Zend Framework 2

Rob Allen
November 2013

Rob Allen

Consultant & ZF trainer
@akrabat
<http://akrabat.com>



What are we going to do?

Look at these key ZF2 features:

- MVC
- Modules
- ServiceManager
- EventManager

... with lots of code!

Getting started

ZFTool

ZFTool is a utility module for maintaining modular ZF2 applications

- Get
<https://packages.zendframework.com/zftool.phar>
- Place in /usr/local/bin
- chmod a+x /usr/loca/bin/zftool.phar

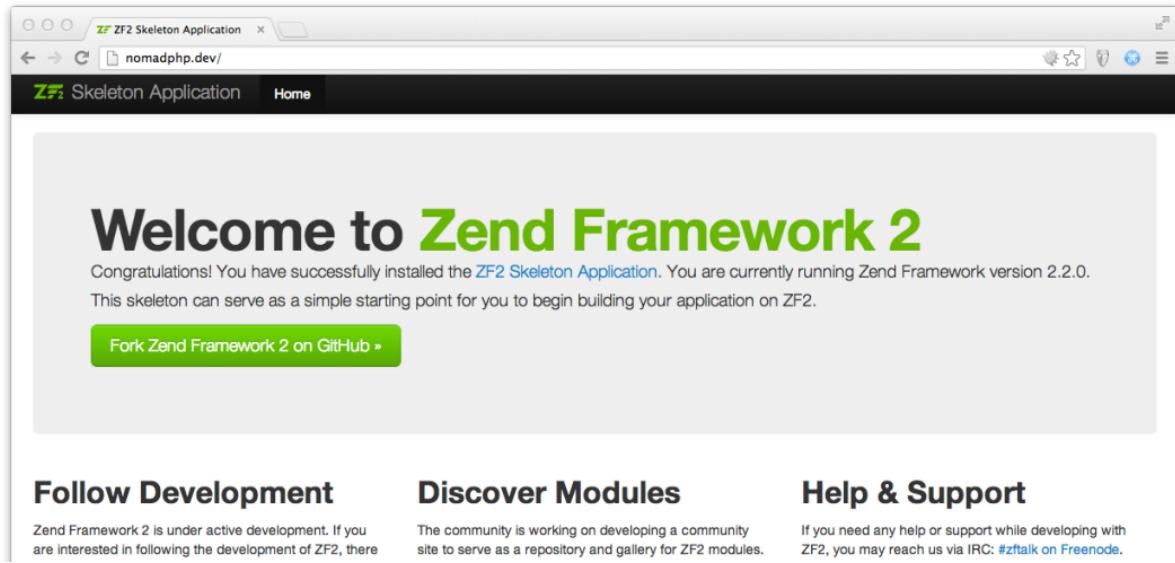
Create a project

```
$ zftool.phar create project nomadphp
ZF2 skeleton application installed in nomadphp.
In order to execute the skeleton application you
need to install the ZF2 library.
Execute: "composer.phar install" in nomadphp
For more info in nomadphp/README.md
```

Composer

```
$ cd nomadphp
$ ./composer.phar install
Loading composer repositories with package inform...
Installing dependencies
- Installing zendframework/zendframework (2.2.0)
  Downloading: 100%
...
Writing lock file
Generating autoload files
```

Skeleton application



The screenshot shows a web browser window with the title bar "ZF ZF2 Skeleton Application". The address bar contains "nomadphp.dev/". The page itself has a dark header with the Zend logo and "Skeleton Application" text, and a "Home" link. The main content area features a large green header "Welcome to Zend Framework 2". Below it, a message says "Congratulations! You have successfully installed the ZF2 Skeleton Application. You are currently running Zend Framework version 2.2.0. This skeleton can serve as a simple starting point for you to begin building your application on ZF2." A green button at the bottom left says "Fork Zend Framework 2 on GitHub »". At the bottom of the page, there are three sections: "Follow Development", "Discover Modules", and "Help & Support".

ZF ZF2 Skeleton Application

nomadphp.dev/

ZF: Skeleton Application Home

Welcome to Zend Framework 2

Congratulations! You have successfully installed the [ZF2 Skeleton Application](#). You are currently running Zend Framework version 2.2.0. This skeleton can serve as a simple starting point for you to begin building your application on ZF2.

Fork Zend Framework 2 on GitHub »

Follow Development

Zend Framework 2 is under active development. If you are interested in following the development of ZF2, there

Discover Modules

The community is working on developing a community site to serve as a repository and gallery for ZF2 modules.

Help & Support

If you need any help or support while developing with ZF2, you may reach us via IRC: [#zftalk on Freenode](#).

Directories & files

```
config
└── autoload
    ├── global.php
    ├── local.php
    └── application.config.php
data
module
└── Application
vendor
└── init_autoloader.php
```

```
public
├── css
├── img
├── js
└── .htaccess
index.php
```

application.config.php

```
return array(
    'modules' => array(                                // MODULES LIST
        'Application',
    ),
    'module_listener_options' => array(
        'module_paths' => array(                        // MODULE PATHS
            './module',
            './vendor',
        ),
        'config_glob_paths' => array(      // GLOBAL CONFIG
            'config/autoload/{,*.}{global,local}.php',
        ),
    ),
)
```

Modules

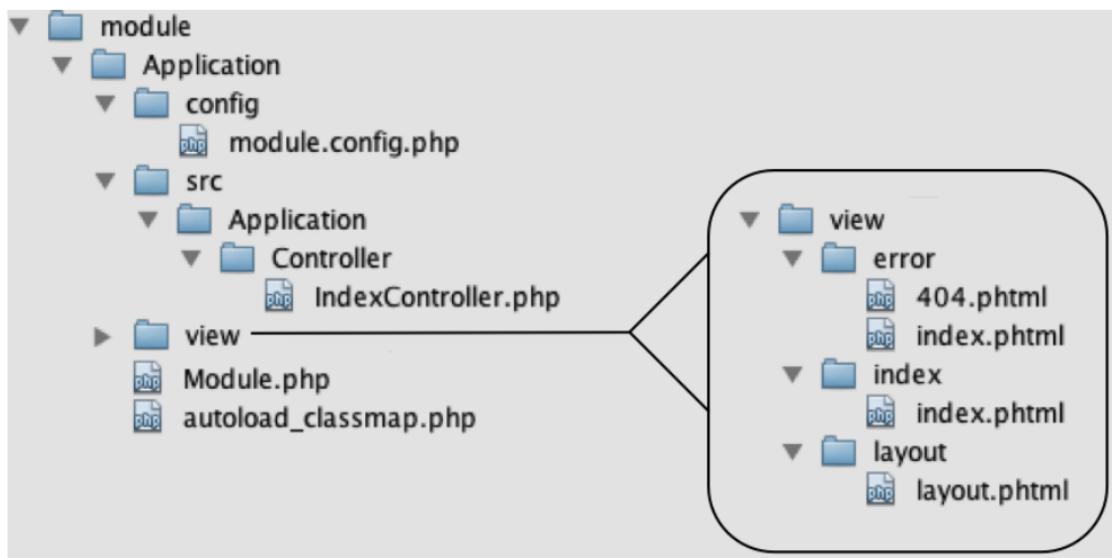
Modules

- Contain all code for the application
 - App-specific in modules/
 - 3rd party in vendor/
- Reusable between applications
 - <http://modules.zendframework.com>

A module...

- is a PHP namespace
- is a class called `Module` within that namespace
 - which provides features to the application
 - autoloading
 - event registration
 - service location
 - has no required structure (but there is convention)

Module directory structure



The Module class

```
namespace Application;

use Zend\Mvc\ MvcEvent;

class Module
{
    public function getAutoloaderConfig() { /* */ }
    public function init() { /* */ }
    public function getConfig() { /* */ }
    public function onBootstrap(MvcEvent $e) { /* */ }

    public function getServiceConfig() { /* */ }
    public function getControllerConfig() { /* */ }
    public function getViewHelperConfig() { /* */ }
    // etc...
}
```

ModuleManager

- Loads all modules
- Triggers events which call specific methods within Module:
 - Autoloading: `getAutoloaderConfig()`
 - Initialisation: `init()`
 - Configuration: `getConfig()`
 - Services: `getServiceConfig()` & friends...
 - Event listener registration: `onBootstrap()`
- Merges all configuration from all modules

Autoloading

```
public function getAutoloaderConfig()
{
    return array(
        'Zend\Loader\ClassMapAutoloader' => array(
            __DIR__ . '/autoload_classmap.php',
        ),
        'Zend\Loader\StandardAutoloader' => array(
            'namespaces' => array(
                __NAMESPACE__ => __DIR__. '/src/'. __NAMESPACE__
            ),
        ),
    );
}
```

(Or just use composer)

Configuration

```
// Application::Module
public function getConfig() {
    return include __DIR__ .'/config/module.config.php';
}

// config/module.config.php:
return array(
    'router' => array( /* ... */ ),
    'translator' => array( /* ... */ ),
    'service_manager' => array( /* ... */ ),
    'controllers' => array( /* ... */ ),
    'view_manager' => array( /* ... */ ),
    // etc...
);
```

Configuration

- Examples use arrays, but can use INI/JSON/etc.
- Configuration is stored in multiple places:
 - A module's `module.config.php` (i.e. result of `getConfig()`)
 - `config/autoload/* .php`
 - Output of ServiceManager related module methods

Configuration merging

Configuration is merged in order:

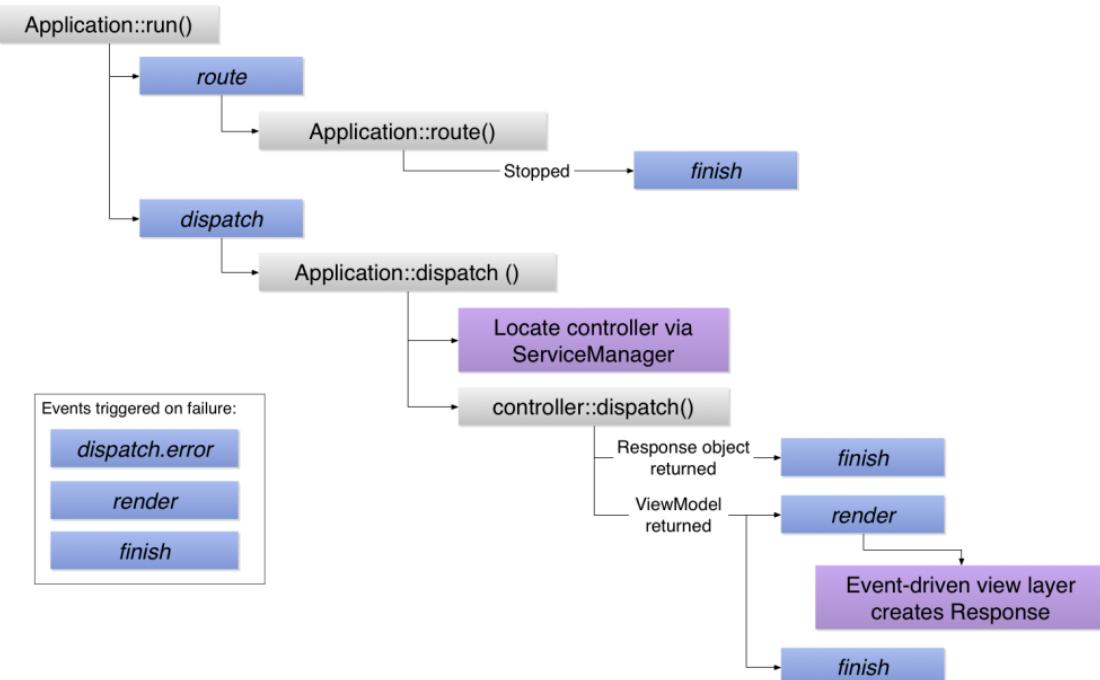
1. All returned arrays from module `getConfig()`
2. All `config/autoload/global.*.php` files
3. All `config/autoload/local.*.php` files
4. All module SM configs: `getServiceConfig()`, etc.

ModuleManager provides caching for steps 1-3.

MVC is event-driven

- All MVC actions are performed by listeners triggered on events:
 - *route*
 - *dispatch*
 - *render*
- You can add your own before or after the MVC ones

MVC event process



Registering events

```
// Application::Module
public function onBootstrap($e)
{
    $app      = $e->getApplication();
    $events   = $app->getEventManager();

    $events->attach('dispatch', function($e) {
        // do something before dispatch
    }, 100);

    $events->attach('dispatch', function($e) {
        // do something after dispatch
    }, -100);
}
```

Zend\ServiceManager

- Dependency Injection Container
- Clean and simple way to configure dependencies.
- Explicit and easy to understand - no magic!
- Used *extensively* by MVC
- More information: <http://bitly.com/bundles/akrabat/1>

Registering services

```
// Application/config/module.config.php:  
return array(  
    'db' => array( /* config */ ),  
  
    'service_manager' => array(  
        'factories' => array(  
            'Zend\Db\Adapter\Adapter'  
                => 'Zend\Db\Adapter\AdapterServiceFactory',  
        ),  
        'invokables' => array(  
            'zfcuser_user_service' => 'User\Service\User',  
        ),  
        'aliases' => array(  
            'my_db' => 'Zend\Db\Adapter\Adapter',  
        ),  
    ),
```

Registering services

```
// Application::Module
public function getServiceConfig()
{
    return array(
        'factories' => array(
            'ErrorHandler' => function($sm) {
                $log = $sm->get('Zend\Log');
                $service = new ErrorHandlingService($log);
                return $service;
            },
        ),
    );
}
```

Routing

Routing

- Inspects URL and matches segments
- Basic unit of routing is the Route
- TreeRouteStack for nested routes
- If matches, returns a RouteMatch object

Types of routes

Name	Example
Literal	/hello
Segment	/news/:year
Regex	/blog/(?<id>[a-zA-Z0-9_-]+)
Wildcard	/*
Hostname	:subdomainexample.com
Scheme	https

RouteStack via config

```
// Gallery/config/module.config.php:  
return array(  
    //...  
  
    'Zend\Mvc\Router\RouteStack' => array(  
        'parameters' => array(  
            'routes' => array(  
                // info about routes go here  
            ),  
        ),  
    ),  
),  
  
//...
```

Route for photo list

```
// Gallery/config/module.config.php, within "routes":  
  
'gallery' => array(                                // ROUTE NAME  
    'type' => 'Literal',  
    'options' => array(  
        'route' => '/photos',                      // URL SEGMENT  
        'defaults' => array(  
            'controller' => 'Gallery\Controller\Photo',  
            'action'      => 'list',  
        ),  
    ),  
    'may_terminate' => true,
```

Name: gallery, Matches: /photos

Nested route for a photo

```
// continuing 'gallery' route...
'child_routes' => array(
    'photo' => array(                                // ROUTE NAME
        'type' => 'Segment',
        'options' => array(
            'route' => '/:id',                      // NEXT URL SEGMENT
            'constraints' => array(
                'id' => '[0-9]+',
            ),
            'defaults' => array(
                'action' => 'image'
            ),
        ),
        'may_terminate' => true,
```

Name: gallery/photo, Matches: /photos/45

Generating URLs

```
// in a controller  
return $this->redirect()->toRoute('gallery');  
  
// in a view script  
<?= $this->url('gallery/photo', array('id' => 45)); ?>
```

Note:

- Use route's name, not URL segment
- Use / for nested route names

Controllers

A controller

```
namespace Gallery;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;

class PhotoController extends AbstractActionController
{
    public function listAction()
    {

    }
}
```

Retrieve parameters

```
public function listAction()
{
    // retrieve param from route match
    $a = $this->params('a');

    // retrieve param from get
    $b = $this->params()->fromQuery('b');

    // retrieve param from post
    $c = $this->params()->fromPost('c');

    // Request object
    $request = $this->getRequest();
}
```

Controller dependencies

```
class PhotoController extends AbstractActionController
{
    protected $mapper;

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

    public function listAction() {
        $photos = $this->mapper->fetchAll();
        return new ViewModel(array(
            'photos' => $photos,
        ));
    }
}
```

Inject dependencies

```
// module/Gallery/Module.php
public function getControllerConfig()
{
    return array(
        'factories' => array(
            'Gallery\Controller\Photo' => function($csm) {
                $sm      = $csm->getServiceLocator();
                $mapper = $sm->get('PhotoMapper');

                return new PhotoController($mapper);
            },
        )));
}
```

(This should look familiar)

Controller response

Action methods can return:

- Response object which is returned immediately
- ViewModel or array of data to pass to the view

Return a Response

```
public function goneAction()
{
    $response = $this->getResponse();
    $response->setStatusCode(410);
    $response->setContent('Gone!');

    return $response;
}
```

Return a ViewModel

```
public function imageAction()
{
    $id = $this->params('id'); // FROM ROUTE

    $image = $this->mapper->load($id);

    $result = new ViewModel(array(
        'photo' => $image,
    ));
    $result->setVariable('name', $image->getName());
    $result->exif = $image->getExif();

    return $result;
}
```

Views

View templates

Set directory location:

```
// in module.config.php
return array(
    // ...
    'view_manager' => array(
        'template_path_stack' => array(
            __DIR__ . '/../view',
        ),
    ),
);
```

default template:

{module}/{controller}/{action}.phtml

Override default template

```
public function listAction()
{
    $result = new ViewModel(array(
        'images' => $this->photoMapper->fetchAll(),
    ));

    $result->setTemplate('photos/list');

    return $result;
}
```

Writing view scripts

- PHP within HTML
- Script extension: .phtml
- Assigned variables accessed via `$this`
- View helpers for reusable complex functionality:

```
<?= $this->escapeHtml($this->photo->title) ?>
```

View script

```
<ul>
<?php foreach($this->photos as $image) :
    $url = $this->url('gallery/photo',
                      array('id' => $image->getId()));
?>
<li>
    <a href="= $url ?&gt;"&gt;
        &lt;?= $this-&gt;displayImage($image) ?&gt;
    &lt;/a&gt;
&lt;/li&gt;
&lt;?php endforeach; ?&gt;
&lt;/ul&gt;</pre
```

Layouts

- For cohesive look and feel
- Includes default CSS, JS & structural HTML
- Default layout template is *layout/layout*
- Implemented as nested view models:
 - “root” view model is the layout
 - Renders action view model

Layout view script

```
<?php
$this->headTitle()->setSeparator(' - ')
->setAutoEscape(false);
$basePath = $this->basePath();
$this->headLink()->appendStylesheet(
    $basePath . '/css/style.css');
$this->headLink(array(
    'rel'  => 'shortcut icon',
    'type' => 'image/vnd.microsoft.icon',
    'href' => $basePath . '/images/favicon.ico',
));
echo $this->doctype();
?>
```

Layout view script (cont)

```
<?= $this->doctype() ?>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <?= $this->headTitle('Photo App') ?>
    <?= $this->headLink() ?>
    <?= $this->headScript() ?>
  </head>
  <body>

    <?= $this->content ?>

    <footer><p>&copy; 2013 Rob Allen</p></footer>
  </body>
</html>
```

View helpers

- When you do the same thing twice in a script
- Interaction with model from view
- Hide complexity from view script
- Usually extend from AbstractHelper
- Implement `__invoke()` for ease of use:

View helper registration

```
// Application/config/module.config.php:  
'view_helpers' => array(  
    'invokables' => array(  
        'date' => 'Application\View\Helper\FormatDate',  
        'formRow' => 'Application\View\Helper\FormRow',  
    ),  
,  
)
```

(This should look familiar)

...or in Module.php

```
public function getViewHelperConfig()
{
    return array(
        'factories' => array(
            'displayChoice' => function($vhsm) {
                $sm = $vhsm->getServiceLocator();
                $mapper = $sm->get('ChoiceMapper');

                return new DisplayChoice($mapper);
            },
        ),
    );
}
```

(This should also look familiar)

A view helper

```
namespace Application\View\Helper;
use Zend\View\Helper\AbstractHelper;

class FormatDate extends AbstractHelper
{
    public function __invoke($date, $format='d M Y')
    {
        $date = strtotime($date);
        $html = $date ? date($format, $date) . '';
        return $html;
    }
}

// use:
<?= $this->date($photo->getDateTaken()); ?>
```

In summary

- *Everything* is in a module
- Extend and modify using events
- Wire up using ServiceManager

Thank you!

Rob Allen : <http://akrabat.com> : @akrabat