FREE eBook

LEARNING zend-framework2

Free unaffiliated eBook created from
Stack Overflow contributors.

#zend-framework2

Table of Contents

About 1

Chapter 1: Getting started with zend-framework2 2

Remarks 2

Examples 2

Installation or Setup 2

Using Composer - Recommended way 2

Using Git Submodules 3

HTTP Server Setup 3

OPTION 1 - PHP CLI Server 3

OPTION 2 - A custom HTTP Server 4

A simple Hello World 4

How to create a factory 5

Chapter 2: Installation 7

Examples 7

installing via composer (github) 7

Chapter 3: Introduction to Zend Expressive 8

Examples 8

A simple Hello World 8

Credits 11

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: zend-framework2

It is an unofficial and free zend-framework2 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official zend-framework2.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with zend-framework2

Remarks

Zend Framework 2 (ZF2) is a modern and flexible PHP framework that helps web developpers to build web applications of different complexities. The major sponsor of company Zend Framework is Zend Technologies, which makes it very strong and stable. There are two major improvments of this second version over ZF1. First, a module-based architecture has been adopted by default without any tweak. This comes handy when developping a big sized web application that requires a decomposition to modules. Second, ZF2 implements all the features PHP5.3+ can offer particularly the namespaces. In the previous versions, a controller class is named as follows:

```
class IndexController extends Zend_Controller_Action
{

}
```

This same class is rewritten in ZF2 as follows:

```
namespace Application\Controller;
use Zend\Mvc\Controller\AbstractActionController;

class IndexController extends AbstractActionController
{

}
```

The following are some other exciting features of ZF2:

- Dependency Injection
- EventManager
- ServiceManager

Examples

Installation or Setup

Detailed instructions on getting Zend Framework 2 set up or installed. There are various ways of installing the framework. Below are some of them:

Using Composer - Recommended way

Assuming `composer` is installed on the target box.

To install a skeleton MVC application, run in your terminal to create a new zend framework 2 project in specified location:

```
php composer.phar create-project -sdev \
    --repository-url="https://packages.zendframework.com" \
    zendframework/skeleton-application path/to/install
```

to manually install a **minimal** ZF2 (Zend MVC + its handful of dependencies), run in your command line:

```
composer require zendframework/zend-mvc
```

or for a **full-fledge**d ZF2 (+64 modules):

```
composer require zendframework/zendframework`
```

Please note that the first option runs an installer that will provide you with a fully-functionnal application along with the usual application directories structure. Other options will let you build the whole application from scratch as it simply provides ZF2 modules to build upon.

Using Git Submodules

Run the command below to clone zf2 and it's dependencies recursively from Github:

```
git clone git://github.com/zendframework/ZendSkeletonApplication.git --recursive
```

HTTP Server Setup

A typical web application requires a running a HTTP service listening a dedicated port (usually :80) to pass incoming requests to application, process and serve the output (response) back.

> Note: You can also write console-aware applications in Zend Framework 2 without a need to a HTTP server.

OPTION 1 - PHP CLI Server

The simplest way to get started if you are using PHP 5.4 or above is to start the internal PHP cli-server in the root directory.

Go to project directory and run:

```
php -S 0.0.0.0:8080 -t public/ public/index.php`.
```

This will start the builtin cli-server on port `8080`, and bind it to all network interfaces.

OPTION 2 - A custom HTTP Server

Configure a virtualhost on Apache *or* Microsoft IIS Server *or* Nginx and pass incoming HTTP requests to the application.

A simple Hello World

In your command line, get in the directory you want to create the project in, then type: `composer create-project zendframework/skeleton-application helloWorldTest`. During installation, you will be asked if you want a minimal install: Let's say yes for the moment, we are just testing.

For simplicity sake, we will use the built-in PHP CLI server. From the command line, get yourself in the root directory of your project (`helloWorldTest`), then run : `php -S 0.0.0.0:8080 -t public/ public/index.php`. Now, open your web browser and go to http://localhost/, you should see the welcome page of the ZF2 Skeleton Application.

If you do so, we will now setup a new page. In `module/Application/config/module.config.php` you can see that a dynamic route is already setup for the application subfolder:

```
return [
    'router' => [
        'routes' => [
            'home' => [
                ...
            ],
            'application' => [
                'type'    => Segment::class,
                'options' => [
                    'route'    => '/application[/:action]',
                    'defaults' => [
                        'controller'    => Controller\IndexController::class,
                        'action'        => 'index',
                    ],
                ],
            ],
        ],
    ],
```

Set a new action "`helloWorldAction()`" in `module/Applicaiton/src/Controller/IndexController.php`:

```
class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        ...
    }

    public function helloWorldAction()
    {
        return new ViewModel();
    }
}
```

Finally, create the view file `module/Application/view/application/index/hello-world.phtml` with the following content:

```
<?php
echo "Hello World !";
```

Now, go to http://localhost/application/hello-world, and say hi to ZF2 !

How to create a factory

When a class needs to be provided with hard dependencies best practice is to use a constructor injection pattern where those dependencies are injected using a factory.

Let's assume that `MyClass` is hard dependent on a value `$dependency` that needs to be resolved from the application config.

```
<?php
namespace Application\Folder;

use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class MyClass
{
    protected $dependency;

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

To inject this dependency a factory class is created. This factory will resolve the dependency from the config and inject the config value on construction of the class and return the result:

```
<?php
namespace Application\Factory;

use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class MyClassFactory implements FactoryInterface
{
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $config = $servicelocator->get('Config');
        $dependency = $config['dependency'];
        $myClass = new MyClass($dependency);
        return $myClass;
    }
}
```

Now that the factory class has been created it has to be registered inside the service manager config in the module config file `module.config.php` under the key factories. It is good practice to use

the same names for both the class and the factory so it is easy to find them in the project folder tree:

```
<?php

namespace Application;

return array(
    //...
    'service_manager' => [
        'factories' => [
            'Application\Folder\MyClass' => 'Application\Factory\MyClassFactory'
        ]
    ],
    //...
);
```

Alternatively the class name constants can be used to register them:

```
<?php

namespace Application;

use Application\Folder\MyClass;
use Application\Factory\MyClassFactory;

return array(
    //...
    'service_manager' => [
        'factories' => [
            MyClass::class => MyClassFactory::class'
        ]
    ],
    //...
);
```

Now the class can be collected at the service manager using the key that we used when registering the factory for that class:

```
 $serviceManager->get('Application\Folder\MyClass');
```

or

```
 $serviceManager->get(MyClass::class);
```

The service manager will find, collect and run the factory and then it returns your class instance with the dependency injected.

Read Getting started with zend-framework2 online: https://riptutorial.com/zend-framework2/topic/1304/getting-started-with-zend-framework2

Chapter 2: Installation

Examples

installing via composer (github)

```
1 cd my/project/dir
2 git clone git://github.com/zendframework/ZendSkeletonApplication.git
3 cd ZendSkeletonApplication
4 php composer.phar self-update
5 php composer.phar install
```

Read Installation online: https://riptutorial.com/zend-framework2/topic/6458/installation

Chapter 3: Introduction to Zend Expressive

Examples

A simple Hello World

Using composer, execute the following command in the directory in which the application will be installed: `composer create-project zendframework/zend-expressive-skeleton expressive-skeleton`.

During installation process, you will be asked to make various decisions.

1. For the default installation question, say no (`n`);
2. For the router, let's use Aura Router (#`1`);
3. For the container, let's use Zend ServiceManager (#`3`);
4. For the template, let's use Zend View (#`3`);
5. Finally, for the error handler, let's use Whoops (#`1`).

Once installed, get yourself in the root directory, `expressive-skeleton`, launch the built-in PHP CLI server: `php -S 0.0.0.0:8080 -t public public/index.php`. Go to http://localhost:8080/ with your browser, your application should now be up and running.

Let's configure a new path to a new middleware. First, open the router config file in `config/autoload/routes.global.php` and add the lines as follow:

```
<?php

return [
    'dependencies' => [
        ...
    ],

    'routes' => [
        [
            'dependencies' => [
                'invokables' => [
                    ...
                ],
                'factories' => [
                    ...
                    // Add the following line
                    App\Action\HelloWorldAction::class => App\Action\HelloWorldFactory::class,
                ],
            ],
        ],
        // Following lines should be added
        [
            'name' => 'hello-world',
            'path' => '/hello-world',
            'middleware' => App\Action\HelloWorldAction::class,
            'allowed_methods' => ['GET'],
        ],
    ],
```

```
];
```

Put the following content in `src/App/Action/HelloWorldFactory.php`:

```
<?php

namespace App\Action;

use Interop\Container\ContainerInterface;
use Zend\Expressive\Template\TemplateRendererInterface;

class HelloWorldFactory
{
    public function __invoke(ContainerInterface $container)
    {
        $template = ($container->has(TemplateRendererInterface::class))
            ? $container->get(TemplateRendererInterface::class)
            : null;

        return new HelloWorldAction($template);
    }
}
```

Then, this content in `src/App/Action/HelloWorldAction.php`:

```
<?php

namespace App\Action;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Zend\Diactoros\Response\HtmlResponse;
use Zend\Diactoros\Response\JsonResponse;
use Zend\Expressive\Template;
use Zend\Expressive\Plates\PlatesRenderer;
use Zend\Expressive\Twig\TwigRenderer;
use Zend\Expressive\ZendView\ZendViewRenderer;

class HelloWorldAction
{
    private $template;

    public function __construct(Template\TemplateRendererInterface $template = null)
    {
        $this->template = $template;
    }

    public function __invoke(ServerRequestInterface $request, ResponseInterface $response,
callable $next = null)
    {
        $data = [];

        return new HtmlResponse($this->template->render('app::hello-world'));
    }
}
```

Then, finally, simply put the following in `templates/app/hello-world.phtml`:

```
<?php echo 'Hello World'; ?>
```

We are done ! Navigate to http://localhost:8080/hello-world, and say "hi" to Zend Expressive !

Read Introduction to Zend Expressive online: https://riptutorial.com/zend-framework2/topic/6109/introduction-to-zend-expressive

Credits

S. No	Chapters	Contributors
1	Getting started with zend-framework2	Community, edigu, Hassan, Sanjeev kumar, Shirraz, Wilt
2	Installation	edigu, Waqar Haider
3	Introduction to Zend Expressive	Shirraz