ZEND EXPRESSIVE FRAMEWORK

for

Web Applications

Version 0.1

Prepared by Thien Kieu

Apr 9, 2019

Table of Contents

Contents

Table of Contents

[Project Configuration 5](#_Toc41058898)

[Project architect and workflow 5](#_Toc41058899)

[Project Directories 6](#_Toc41058900)

[Source 6](#_Toc41058901)

[Test 7](#_Toc41058902)

[Project Dependencies 8](#_Toc41058903)

[Logger 10](#_Toc41058904)

[Logging Configuration Files 10](#_Toc41058905)

[Use LoggerService to log in project 10](#_Toc41058906)

[Log to another destination 11](#_Toc41058907)

[Database 11](#_Toc41058908)

[Doctrine ODM Configuration Files 11](#_Toc41058909)

[Define factory to resolve Document Management 11](#_Toc41058910)

[Add dependence for service manager in dependencies.global.php 12](#_Toc41058911)

[User doctrine document manager 12](#_Toc41058912)

[Authentication**:** 12](#_Toc41058913)

[Router 13](#_Toc41058914)

[Dependence Injection 14](#_Toc41058915)

[Validation 15](#_Toc41058916)

[Case study 17](#_Toc41058917)

[Init new project 17](#_Toc41058918)

[Clone source code 17](#_Toc41058919)

[Configuration db 17](#_Toc41058920)

[Install dependence 18](#_Toc41058921)

[Run local server with wampserver 18](#_Toc41058922)

[Add this line to host file 18](#_Toc41058923)

[Add new module to project 19](#_Toc41058924)

[Create project folder structure like bellow 19](#_Toc41058925)

[Add ConfigProvider.php to project configuration 19](#_Toc41058926)

[Add routes to app 19](#_Toc41058927)

[Add module to autoload. 19](#_Toc41058928)

[Create new a endpoint to show helloworld 20](#_Toc41058929)

[Create new handler file 20](#_Toc41058930)

[Add route to module: 21](#_Toc41058931)

[Declare your handler to container 21](#_Toc41058932)

[Problems you can face 21](#_Toc41058933)

[Create new a endpoint to save request data to database 22](#_Toc41058934)

[Workflow of request 22](#_Toc41058935)

[Implement to handle request 23](#_Toc41058936)

[Working with database**:** 27](#_Toc41058937)

[Add new config value 28](#_Toc41058938)

[App config & module config 28](#_Toc41058939)

[Dev config 29](#_Toc41058940)

[Add new convert request to DTO 29](#_Toc41058941)

[Add new convertor 30](#_Toc41058942)

[Notes 30](#_Toc41058943)

[Register your convertor 31](#_Toc41058944)

[Add new convert DTO to Document 32](#_Toc41058945)

[Add new convertor 32](#_Toc41058946)

[Notes 33](#_Toc41058947)

[Register your convertor 33](#_Toc41058948)

[To get document object from DTO use bellow code 34](#_Toc41058949)

[Add new convertor Document to DTO 34](#_Toc41058950)

[Add new adapter 34](#_Toc41058951)

[Notes 35](#_Toc41058952)

[Register your convertor 36](#_Toc41058953)

[To get DTO object from Document use bellow code 36](#_Toc41058954)

[Add new dependence to project 36](#_Toc41058955)

[Integrate SSO login with other site. 36](#_Toc41058956)

[Deploy product 36](#_Toc41058957)

[Exception handle 36](#_Toc41058958)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Thien Kieu | Apr 09th, 2019 | - Initial version | 0.1 draft |
|  |  |  |  |

This document describes how to create a web-base application using Zend Expressive Framework. You can find out how to structure, configure a project and apply some best programming practices into it.

Technologies used in the framework include:

Doctrine ODM

Zend Expressive 3.0.1

zend-expressive-authentication-oauth2

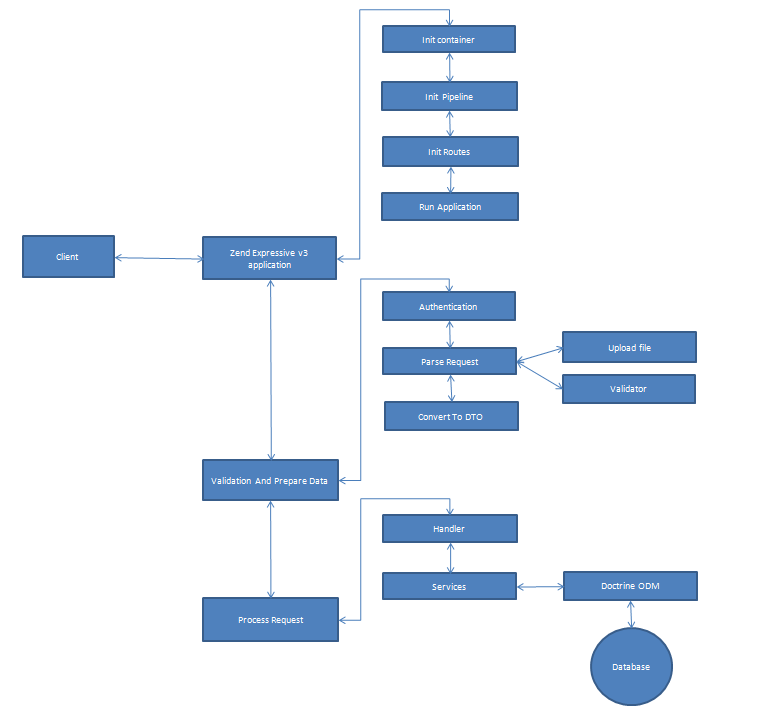
JSON

Validation

Zend Log

# Project Configuration

## Project architect and workflow



## Project Directories

### Source

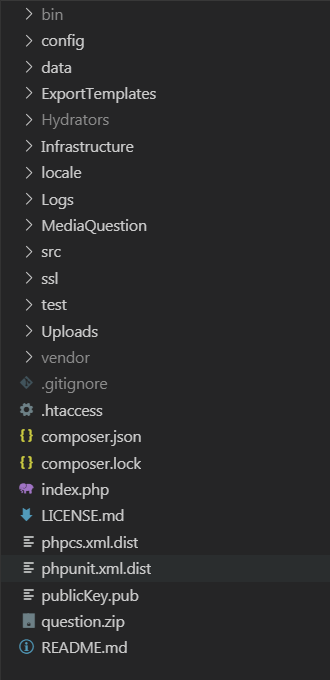
* src – PHP Source code packages and classes
* vendor – Zend Expressive and third party package.
* config – Project configuration (DI, Database configuration, Router)
* Infrastructure - PHP source support to process that don't relate to business.
* ExportTemplate - Template to export Excel
* locale - Translate message for each language such as English, Vietnamese,
* Logs - log files
* .htaccess: pre-config for apache.
* index.php: main file.

### Test

* test – Test Source code packages and classes

Project Structure Example

|  |  |
| --- | --- |
|  |  |



## Project Dependencies

One of the challenges of enterprise build tools is managing transitive dependencies.

In this document use composer to managing dependencies.

Download composer at link: https://getcomposer.org/ then install it. Make sure you have add composer path and PHP path to PATH that allow you run composer in command line.

To add new dependence to project, open command line in project folder and using command:

composer require {your package}

ex:

composer require zendframework/zend-log

To remove package from project using command

composer remove {your package}

ex:

composer remove zendframework/zend-log

Please refer https://getcomposer.org/doc/ to more detail.

Note:

* USE require section to control package for production environment
* USE require-dev section to control package for development environment

Example:

"require": {

"php": "^7.1",

"alcaeus/mongo-php-adapter": "^1.1",

"doctrine/mongodb-odm": "^1.2",

"doctrine/orm": "2.6.3",

"zendframework/zend-component-installer": "^2.1.1",

"zendframework/zend-config-aggregator": "^1.0",

"zendframework/zend-diactoros": "^1.7.1 || ^2.0",

"zendframework/zend-expressive": "^3.0.1",

"zendframework/zend-expressive-authentication-oauth2": "^1.1",

"zendframework/zend-expressive-fastroute": "^3.0",

"zendframework/zend-expressive-helpers": "^5.0",

"zendframework/zend-log": "^2.10",

"zendframework/zend-servicemanager": "^3.3",

"zendframework/zend-stdlib": "^3.1",

"zendframework/zend-validator": "^2.12"

},

"require-dev": {

"phpunit/phpunit": "^7.0.1",

"roave/security-advisories": "dev-master",

"squizlabs/php\_codesniffer": "^2.9.1",

"zendframework/zend-expressive-tooling": "^1.0",

"zfcampus/zf-development-mode": "^3.1",

"filp/whoops": "^2.1.12"

},

## Logger

In this project we log in to file and DB.

For log into file we use Zend-Log: Please refer link [https://zendframework.github.io/zend-log/](https://zendframework.github.io/zend-log/%20) for more detail

For DB log we build API at NodeJS server and use Winston to login into MongoDB

### Logging Configuration Files

Add Zend-Log configuration dependence to project: Open file config.php in {project-folder}/config and add flow item into ConfigAggregator section

    \Zend\Log\ConfigProvider::class,

Add log file configuration.

'log' => [

    'writers' => [

        [

            'name' => 'stream',

            'options' => [

                'stream' => $dir.'/../Logs/'.date('Y-m-d').'.txt',

                'filters' => [

                    'allMessages' => [

                        'name' => 'priority',

                        'options' => [

                            'operator' => '>=',

                            'priority' => \Zend\Log\Logger::EMERG,

                        ]

                    ],

                ],

            ]

        ]

    ]

],

**Enable or disable logger to file or database**

'TRACKING\_CONNECT' => [

        'latestDisConnectURL' => 'http://localhost:3000/getLatestDisconenct',

        'writeLogURL' => 'http://localhost:3000/writeLog',

        'enableLogFile' => true,

        'enableLogRemote' => true,

    ],

Use LoggerService to log in project.

$logger = $this->container->get(\Infrastructure\Services\Interfaces\LogInterface::class);

$logger->writeLog($e);

### Log to another destination

If you want to log to another destination as Database, Syslog, you can change writer. Zend support some writer https://zendframework.github.io/zend-log/writers/. Or you can add new custom for yourself by implement interface Zend\Log\Writer\AbstractWriter

## Database

We use Doctrine ODM to working with nonSQL. Please refer <https://www.doctrine-project.org/projects/doctrine-mongodb-odm/en/2.0/index.html> for more detail.

### Doctrine ODM Configuration Files

'nonsqldb' => [

'mongodb-connection' => 'mongodb://mlab.com:43963/thienkieu', 'document-path' => [

$dir.'/App/src/Documents',

$dir.'/Test/src/Documents'

],

'proxy-path' => $dir.'/Proxies',

'hydrators-path' => $dir.'/../Hydrators',

'dbname' => 'thienkieu',

]

]

Define factory to resolve Document Management.

class DoctrineODMFactory

{

public function \_\_invoke(ContainerInterface $container) : DocumentManager

{

$appConfig = $container->get('config');

$dbConfig = $appConfig['nonsqldb'];

$client = new Client($dbConfig['mongodb-connection'], [], ['typeMap' => ['root' => 'array', 'document' => 'array']]);

$connection = new Connection($client);

$config = new Configuration();

$config->setProxyDir($dbConfig['proxy-path']);

$config->setProxyNamespace('Proxies');

$config->setHydratorDir($dbConfig['hydrators-path']);

$config->setHydratorNamespace('Hydrators');

$config->setDefaultDB($dbConfig['dbname']);

$config->setMetadataDriverImpl(AnnotationDriver::create($dbConfig['document-path']));

AnnotationDriver::registerAnnotationClasses();

$dm = DocumentManager::create($connection, $config);

return $dm;

}

}

Add dependence for service manager in dependencies.global.php**.**

'documentManager' => App\Factory\DoctrineODMFactory::class

User doctrine document manager**.**

$dm = $this->container->get('documentManager');

$builder = $dm->createQueryBuilder(array(\App\Documents\ReadingSection::class));

$builder = $builder->field('questions.content')->equals(new Regex(‘content’, 'i')

$query = $builder->getQuery();

$documents = $query->execute();

Authentication**:**

To access system, client must add token with Authentication header. Access url /token get token with bellow parameter.

{

"grant\_type": "sso",

"userInfo": "",

"client\_id": "client\_test",

"client\_secret": "test",

"scope": "test"

}

Parameter to get token for doExam:

{

        grant\_type: "pin",

        PIN: pin,

        client\_id: "client\_test",

        client\_secret: "test",

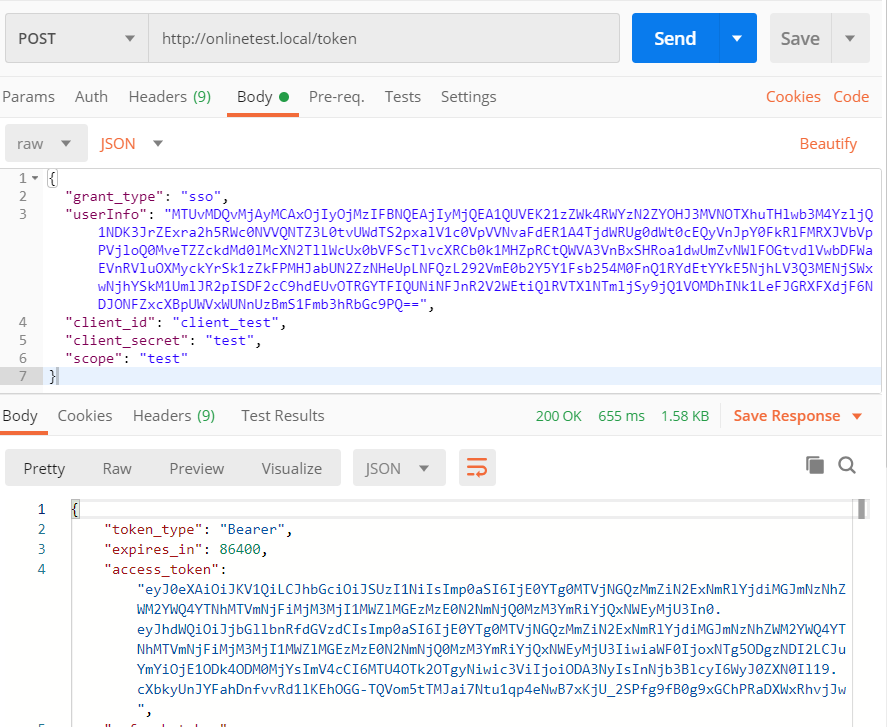
        scope: "test"

};

Method: POST

Format: JSON

Ex:



Notes:

Provide grant\_type is sso to get token that access training coordinator site.

Prodive grant type is `pin` to get token that access do exam site.

userInfo: This info will be return after login on CRM.

## Router

Router is spliced into modules. In each module have routers.php file that contain all router of this module.

Router have three part:

Path url:

Hander: when user access url this handler will be called.

Name: this is name of router

URL method: PUT, GET, POST …

Ex:

 $app->get('/coordinator/question/export',

            Test\Handlers\ExportQuestionHandler::class,

            'question.export'

          );

$app->post('/coordinator/questions/create',

            Test\Handlers\CreateQuestionHandler::class,

            'questions.create'

          );

All router will be restrict for access, you must provide token on Authentication header to access. In some case you don’t want to restrict access, please add your router in authenticationExcludeUrl section in app.global.php in {project folder}/config

In case integrated with other, you can assign special token for that system, please add this token in authenticationExcludeToken section in app.global.php in {project folder}/config

Ex:

    'authenticationExcludeToken' => [

        'Bearer CRMbackend'

    ],

## Dependence Injection

Use Zend\ServiceManager to support dependence Injection. Each module have ConfigProvide.php to config dependence injection for that module. Zend\ServiceManager provide us Factory, Alias, Invokable… Please refer <https://docs.zendframework.com/zend-servicemanager/quick-start/> for more details.

Zend\ServiceManger supported factory to resolve service with default service have default constructor (non parameter). To resolve service that have more parameter on constructor, you must implement your factory.

In this project, we have build some basic factory to resolve service and handler:

- Infrastructure\Factory\ServiceFactory: That will resolve service that have multiple implement.

When service manager resolve service, it will call isHandler of each implement and pass options to service, if service want to handle it will return true or vice versa.

- Infrastructure\Factory\BaseFactory: That will resolve service that and pass ServiceManager and Option to service.

Ex:

We have DataParserInterface interface and have 2 services implement this interface ExcelParserService and WordParserService.

We must declare to service manager know this interface have 2 implement by add the following lines to dependencies.global.php.

\Infrastructure\DataParser\DataParserInterface::class => [

            \Infrastructure\DataParser\ExcelParserService::class,

            \Infrastructure\DataParser\WordParserService::class

]

ExcelParserService hasn’t default contructor, so we must identity what factory will resolve this service. In this case, use baseFactory

Infrastructure\DataParser\ExcelParserService::class =>

            Infrastructure\Factory\BaseFactory::class,

To get this service just get it from ServiceManager.

 $this->dataParser = $this->container->get(DataParserInterface::class,

         [DataParserInterface::FileTypeKey => 'excel']);

The second parameter will be pass to IsHanlder of service. If service return true, ServiceManager will create new instance and return to caller service. The instance of service will be add to cache and will be return for other time.

To get new service, without from cache use build method.

 $this->dataParser = $this->container->build(DataParserInterface::class,

         [DataParserInterface::FileTypeKey => 'excel']);

## Validation

In Project we support validate request data before convert it to DTO object. Validate request will be config at {Project folder}/Config/autoload/adapter.global.php

Ex:

'validatorRequestAdapters' => [

        //\Test\Validator\CreateReadingSectionValidatorAdapter::class,

        \Test\Validator\CreateQuestionValidatorAdapter::class,

        \Test\Validator\CreateTestWithSectionValidatorAdapter::class,

        \Test\Validator\CreateExamWithSectionValidatorAdapter::class,

        \Test\Validator\EnterPinValidatorAdapter::class,

        \Test\Validator\CreateSourceValidatorAdapter::class,

        \Test\Validator\UpdateMarkValidatorAdapter::class,

    ],

Validate middleware call isHandleValid($routerName, $request) of validate. Validate return true to validate this request.

Ex:

public function isHandleValid($routerName, $request) : bool

    {

        $body = $request->getParsedBody();

        $type = isset($body->type) ? $body->type: '';

        if (($routerName === \Config\AppRouterName::CreateQuestion || $routerName === \Config\AppRouterName::UpdateQuestion)

             &&

            ($type === DTOName::Reading || $type === DTOName::Listening )

        ) {

            return true;

        }

        return false;

    }

I18N:

In this project we use Zend\I18n\Translator\Translator to support for I18n. Please refer <https://framework.zend.com/manual/2.1/en/modules/zend.i18n.translating.html> for more details.

To translate your message, use this code.

 $translator = $this->container->get(\Config\AppConstant::Translator);

 $translator->translate('There is not convertor, Please check it again');

Add translate file is located at {Project-folder}/locale

Add your translate to en\_US to translate to English language:

'There is not convertor, Please check it again' =>

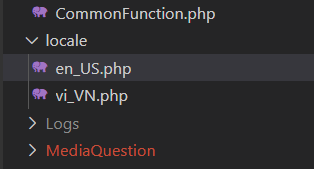
'There is not convertor. Please check it again.',

add your translate to vi\_VN to translate your message to Vietnames.

'There is not convertor, Please check it again' =>

'Không có chuyển đổi. Vui long kiểm tra lại.',

Your can add more file to translate to other language.



# Case study

## Init new project

Clone source code**.**

move to project folder, open command line and type command git clone {your repo}.

Configuration db**:**

In file app.config in config/autoload folder.

if your project use NoSQL database, Please update config in section nosqldb

'nonsqldb' => [

        'mongodb-connection' => 'mongodb://192.168.191.79:27017/demo\_2608',

        'document-path' => [

            $dir.'/App/src/Documents',

            $dir.'/Test/src/Documents',

            $dir.'/ODMAuth/src/Documents',

        ],

        'proxy-path' =>  $dir.'/Proxies',

        'hydrators-path' =>  $dir.'/../Hydrators',

        'dbname' => 'demo\_2608',

    ],

mongodb-connection: url connect to database, it dependence your database.

document-path: Document folders

proxy-path: Proxy folder

hydrators-path: Hydrators folder,

dbname: name of your database.

**if your project use SQL database, please update config in section sqldb**

'sqldb' => [

        'db' => [

            'driver'   => 'pdo\_mysql',

            'user'     => 'root',

            'password' => '',

            'dbname'   => 'onlinetest',

        ],

        'entity-path' => [

            $dir.'/App/src/Entity'

        ],

    ],

driver: driver connect to sql server

user:

password

dbname:

entity-path: Entity folder.

**Note**: If you want to config online for dev or only for local, please update these info at config/development.config file.

Install dependence**:**

Open command line at project, type composer install and waiting for composer install.

Run local server with wampserver:

Download and install wampserver, please refer this release: [https://blog.templatetoaster.com/how-to-install-wamp](https://blog.templatetoaster.com/how-to-install-wamp/)

Copy this code and pass in httpd-vhosts.conf

<VirtualHost backend.onlinetest.com >

ServerName backend.onlinetest.com

ServerAlias backend.onlinetest.com

DocumentRoot "D:/projects/OnlineTest/NewOnlineTest"

<Directory "D:/projects/OnlineTest/NewOnlineTest/">

Options +Indexes +Includes +FollowSymLinks +MultiViews

AllowOverride All

Require local

</Directory>

</VirtualHost>

Replace backend.onlinetest.com with your domain

Replace D:/projects/OnlineTest/NewOnlineTest with your project path

Replace "Require local" to Require local "Require all granted" if you want access via IP or from another machine.

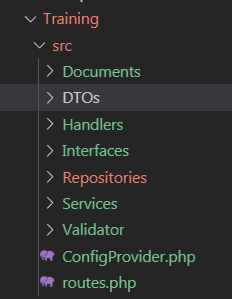
Add this line to host file**:**

127.0.0.1 backend.onlinetest.com

Replace backend.onlinetest.com with your domain

## Add new module to project

Create project folder structure like bellow**:**



For sample you can duplicate Training module and replace to your module,

Open file ConfigProvider.php and update namespace to your module

### Add ConfigProvider.php to project configuration

Open file config.php at config folder and pass this line to config file (rename namespace for your module)

    Training\ConfigProvider::class,

ex:

    // Default App module config

    App\ConfigProvider::class,

    Test\ConfigProvider::class,

    ODMAuth\ConfigProvider::class,

    Training\ConfigProvider::class,

### Add routes to app

open file routes in config/routes.php and pass this line

(require $dir.'/Training/src/routes.php')($app, $factory, $container);

ex:

    (require $dir.'/App/src/routes.php')($app, $factory, $container);

    (require $dir.'/ODMAuth/src/routes.php')($app, $factory, $container);

    (require $dir.'/Test/src/routes.php')($app, $factory, $container);

    (require $dir.'/Training/src/routes.php')($app, $factory, $container);

### Add module to autoload.

open composer.json and the follow line to section autoload

            "Training\\": "src/Training/src/",

ex:

    "autoload": {

        "psr-4": {

            "App\\": "src/App/src/",

            "Training\\": "src/Training/src/",

            "Test\\": "src/Test/src/",

            "Infrastructure\\": "Infrastructure",

            "Config\\": "config",

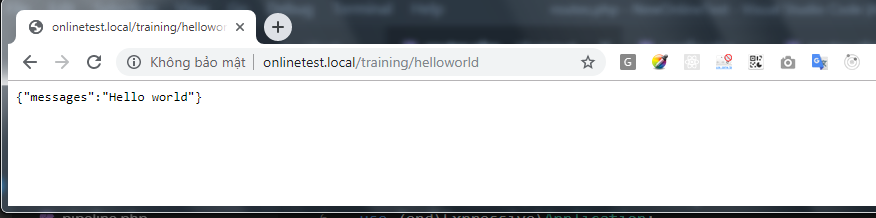
            "ODMAuth\\": "src/ODMAuth/src/"

        }

    },

run composer dump-autoload in comamd line to update autoload.

## Create new a endpoint to show helloworld



### Create new handler file

In Handlers folder create new HelloworldHandler .php and pass this code to your handler

<?php

declare(strict\_types=1);

namespace Training\Handlers;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\RequestHandlerInterface;

use Psr\Container\ContainerInterface;

use Zend\Diactoros\Response\JsonResponse;

class HelloWorldHandler implements RequestHandlerInterface

{

    /\*\* @var Psr\Container\ContainerInterface \*/

    private $container;

    public function \_\_construct(ContainerInterface $container) {

        $this->container = $container;

    }

    public function handle(ServerRequestInterface $request) : ResponseInterface

    {

        return new JsonResponse([

            'messages' => 'Hello world',

        ]);

    }

}

### Add route to module:

open file routes.php in YourModule/src and add new route with format

$app->{method\_name: get, post..}('your path', 'Handler','route name');

ex:

$app->get('/training/helloworld', Training\Handlers\HelloWorldHandler::class, 'training.helloword');

We can explain this line at bellow:

When have request with path /training/helloworld and get method, system will call Training\Handlers\HelloWorldHandler to process it, and we named it 'training.helloword'

Declare your handler to container

Zend expressive support default factory that will resolve class there isn't argument in constructor. If your constructor have more argument, you must declare what factory will resolve your class.

Open file ConfigProvider.php in YourModule/src add new line to factories:

Handlers\HelloWorldHandler::class => \Infrastructure\Factory\BaseFactory::class,

This line can be explain: When system need a instance of class Handlers\HelloWorldHandler then \Infrastructure\Factory\BaseFactory will init instance and return it for system.

ex:

public function getDependencies() : array

    {

        return [

            'invokables' => [

            ],

            'factories'  => [

                Handlers\HelloWorldHandler::class => \Infrastructure\Factory\BaseFactory::class,

            ],

            'aliases' => [

                //Services\SectionServiceInterface::class => Services\SectionService::class,

            ],

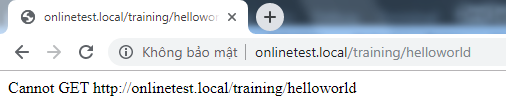
        ];

    }

Time to see a result, enter your path on browser and you will see hello world message.

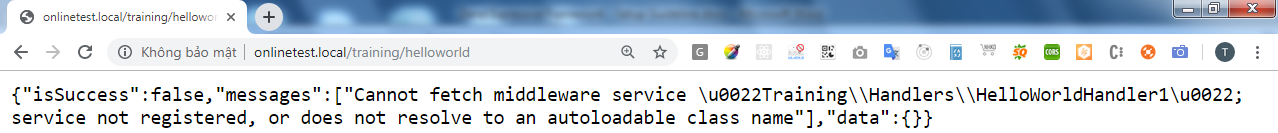
### Problems you can face

#### Cannot found URL



if you face this problem, please check route config, make sure you correct method (post, get, put ...) and url (yourdomain/path);

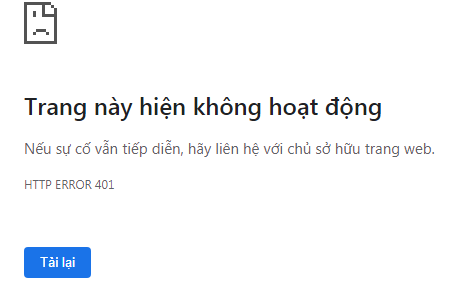
#### Cannot resolve service or class



if you face this problem, please check handler and declare to resolve handler,

make sure correct handler and namespace, make sure you added declaretion in ConfigProvider.php

#### Unauthorization 401



All request to system is require token, if you face this problem that mean you theren't permission to access you must add token into Authentication to header of request,

If you want to ignore authentication for this url, you need to add route name info getAuthenticationExcludeUrl in ConfigProvider.php

    public function getAuthenticationExcludeUrl(): array

    {

        return [

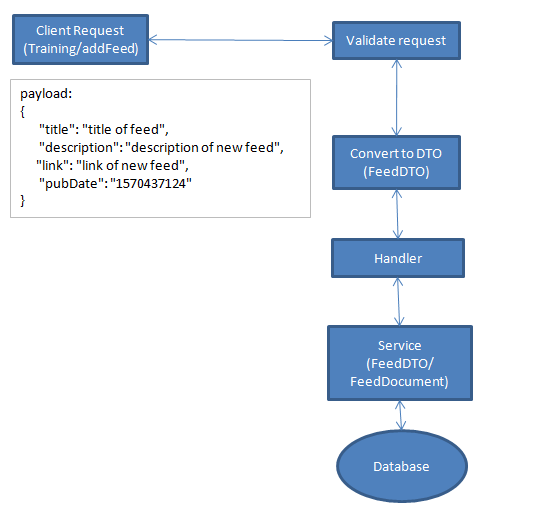
           'training.helloword',

        ];

    }

## Create new a endpoint to save request data to database

### Workflow of request



### Implement to handle request

Add new hanlder, Declare your hanlder with serviceManager, Add router to module. Please refer to create new endpoint to show helloword.

#### Add new validation

In validator folder add new FeedValidator.php and code like bellow:

<?php

declare(strict\_types=1);

namespace Training\Validator;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Infrastructure\Validator\ValidatorAdapterInterface;

class FeedValidatorAdapter implements ValidatorAdapterInterface

{

    protected $container;

    /\*\*

     \* Class constructor.

     \*/

    public function \_\_construct($container)

    {

        $this->container = $container;

    }

    public function isHandleValid($routerName, $request) : bool

    {

        if ($routerName === \Config\AppRouterName::Training\_AddFeed) {

            return true;

        }

        return false;

    }

    public function valid(ServerRequestInterface $request, ResponseInterface & $messageResponse): bool

    {

        return true;

    }

}

In which:

isHandleValid method receive current matching route name, and request object (contain all info of client request such as header, data user submit ...). This method will be call to check whether this validator handle this request or not, return true If this this validator will validate this request.

valid: the place to put code validate request. This will be call to if isHandleValid return true.

Add sample validate to your validator.

public function valid(ServerRequestInterface $request, ResponseInterface & $messageResponse): bool

    {

        $isError = false;

        $body = $request->getParsedBody();

        if (!property\_exists($body, 'title')) {

            $isError = true;

            $errors = [

                $translator->translate('Title is required.')

            ];

        }

        return $isError;

    }

just require title field.

Feel free to validate here. Zend support some validator, please refer link to get more detiail.

#### Declare your validate to container

Open ConfigProvider.php in your module, in getValidatorRequestAdapters function add this line to return value

            \Training\Validator\FeedValidatorAdapter::class,

ex:

    public function getValidatorRequestAdapters() {

        return [

            \Training\Validator\FeedValidatorAdapter::class,

        ];

    }

#### Mapping request data to DTO

add new DTO class: In DTOs folder create new FeedDTO.php file and pass this code to your DTO class

<?php

declare(strict\_types=1);

namespace Training\DTOs;

class FeedDTO implements \JsonSerializable

{

    protected $title;

    protected $description;

    protected $link;

    protected $pubDate;

    public function jsonSerialize() {

        $ret = new \stdClass();

        $ret->description = $this->getDescription();

        $ret->link = $this->getLink();

        $ret->pubDate = $this->getPubDate();

        $ret->title = $this->getTitle();

        return $ret;

    }

}

Note: you must get set/get for each property.

Declare to system to convert your request data to DTO object.

    public function getRequestToDTOClass() {

        return [

            'training.helloword'  => \Training\DTOs\FeedDTO::class,

        ];

    }

this line mean, the data of request have route 'training.helloworld' will be convert to FeedDTO object. If your data to complex or you want to custom convertor, please refer to how to create convertor case study.

#### Create new handler and declare handler

please refer to create helloword endpoint.

update your handler code like this:

    public function handle(ServerRequestInterface $request) : ResponseInterface

    {

        $dto = $request->getAttribute(\Config\AppConstant::DTODataFieldName);

        $feedService = $this->container->get(FeedServiceInterface::class);

        $ok = $feedService->addNewFeed($dto, $messages);

        return new JsonResponse([

            'isSuccess' => $ok,

            'messages' => $messages

        ]);

    }

We use get(FeedServiceInterface ::class) method of container (ServiceManager) to get instance of FeedService replace for new FeedService directly then we call this service to add new feed and return response to client.

#### Create service to management feed data

In Interfaces folder create new FeedServiceInterface.php as bellow.

<?php

namespace Training\Interfaces;

interface FeedServiceInterface {

    public function addNewFeed(\Training\DTOs\FeedDTO $feedDTO, & $messages);

    public function editFeed(\Training\DTOs\FeedDTO $feedDTO, & $messages);

    public function deleteFeed(\Training\DTOs\FeedDTO $feedDTO, & $messages);

}

In Services folder create new FeedService.php implement FeedServiceInterface as bellow

<?php

declare(strict\_types=1);

namespace Training\Services;

use Zend\Log\Logger;

use Infrastructure\Convertor\DTOToDocumentConvertorInterface;

use Infrastructure\Convertor\DocumentToDTOConvertorInterface;

use Infrastructure\Interfaces\HandlerInterface;

class FeedService implements Interfaces\FeedServiceInterface, HandlerInterface

{

    protected $container;

    protected $dm;

    protected $options;

    public function \_\_construct($container, $options) {

        $this->container = $container;

        $this->options = $options;

        $this->dm = $this->container->get(\Config\AppConstant::DocumentManager);

    }

    public function isHandler($dto, $options = []) {

        return true;

    }

}

In which:

$container is ServiceContainer, using this container to get services or other object we need.

$dm is DocumentManager of doctrine ODM, using this object to working with database.

Declare your service to dependence injection

    public function getResolveService() {

        return [

            \Training\Interfaces\FeedServiceInterface::class => [

                \Training\Services\FeedService::class,

            ],

        ];

    }

This line mean, Everywhere we need FeedServiceInterface object, system will return instance of FeedService and return it for us.

We add code to save feed to database

public function addNewFeed(\Traing\DTOS\FeedDTO $feedDTO, & $messages) {

        $messages = [];

        try{

            $dtoToDocumentConvertor = $this->container->get(DTOToDocumentConvertorInterface::class);

            $document = $dtoToDocumentConvertor->convertToDocument($feedDTO);

            $this->dm->persist($document);

            $this->dm->flush();

            return true;

        } catch(\Exception $e){

            $messages[] = $e->getMessage();

            $logger = $this->container->get(Logger::class);

            $logger->info($e);

            return false;

        }

    }

Working with database**:**

Create new document to mapping with Database

In Documents folder create new FeedDocument.php document.

<?php

namespace Training\Documents;

use Doctrine\ODM\MongoDB\Mapping\Annotations as ODM;

/\*\*

 \* @ODM\Document(collection="feeds",

 \* repositoryClass \Training\Repositories\FeedRepository")

 \*/

class FeedDocument

{

  /\*\* @ODM\Id \*/

  protected $id;

  /\*\* @ODM\Field(type="string") \*/

  protected $description;

  /\*\* @ODM\Field(type="string") \*/

  protected $title;

  /\*\* @ODM\Field(type="string") \*/

  protected $link;

  /\*\* @ODM\Field(type="date") \*/

  protected $pubDate;

  /\*\* @ODM\Field(type="string") \*/

  protected $guid;

}

Note: Please add get/set method for each property.

collection = "document name in your database"

repositoryClass = "path to your custom repository class". If you don't write customize query, you can ignore this option.

Add new repository

<?php

declare(strict\_types=1);

namespace Training\Repositories;

use Doctrine\ODM\MongoDB\DocumentRepository;

class FeedRepository extends DocumentRepository

{

}

## Add new config value

### App config & module config

We can add config in app.global.php or in ConfigProvider.php in each module.

The config is don’t relation module should push at app.global.php and the config release module should put on that module in ConfigProvider.php

Ex:

DB config should put on app.global.php

'nonsqldb' => [

        'mongodb-connection' => 'mongodb://192.168.191.79:27017/demo\_2608',

        'document-path' => [

            $dir.'/App/src/Documents',

            $dir.'/Test/src/Documents',

            $dir.'/ODMAuth/src/Documents',

        ],

        'proxy-path' =>  $dir.'/../Proxies',

        'hydrators-path' =>  $dir.'/../Hydrators',

        'dbname' => 'demo\_2608',

    ],

Validation adapter is related to module should put on that module.

public function getValidatorRequestAdapters(): array {

        return [

            //\Test\Validator\CreateReadingSectionValidatorAdapter::class,

            \Test\Validator\CreateQuestionValidatorAdapter::class,

            \Test\Validator\CreateTestWithSectionValidatorAdapter::class,

            \Test\Validator\CreateExamWithSectionValidatorAdapter::class,

            \Test\Validator\EnterPinValidatorAdapter::class,

            \Test\Validator\CreateSourceValidatorAdapter::class,

            \Test\Validator\UpdateMarkValidatorAdapter::class,

        ];

    }

### Dev config

In case you want to override these setting on development environment, put it on development.config.php

Ex: Override Db config

'nonsqldb' => [

        'mongodb-connection' => 'mongodb://localhost:27017/onlinetest',

        //'mongodb-connection' => 'mongodb://thienkieu:Mlab0958588127@ds253017.mlab.com:53017/onlinetest\_demo?retryWrites=false',

        'document-path' => [

            $dir.'/App/src/Documents',

            $dir.'/Test/src/Documents',

            $dir.'/ODMAuth/src/Documents',

        ],

        'proxy-path' =>  $dir.'/../Proxies',

        'hydrators-path' =>  $dir.'/../Hydrators',

        'dbname' => 'onlinetest',

    ],

## Add new convert request to DTO

Data sent from client will be convert to object. If your data have complex structure you can add convert adapter to convert request data to your DTO

### Add new convertor

Ex: Create adapter to convert writing request data to WritingDTO object and pass it to writing handler.

In {Project-folder}/{your-module}/src/convertor/DTO add new filde ToWriringDTOAdapter and add code like bellow:

<?php

declare(strict\_types=1);

namespace Test\Convertor\Adapter\DTOs;

use Infrastructure\Convertor\ToDTOAdapter;

use Test\Enum\DTOName;

use Config\AppConstant;

class ToWritingDTOAdapter extends ToDTOAdapter {

    public function isHandleConvertToDTO($dtoObject, $options = []) : bool

    {

        $type = isset($dtoObject->type) ? $dtoObject->type: '';

        if (isset($options[\Config\AppConstant::DTOKey]) && $options[\Config\AppConstant::DTOKey] === DTOName::QuestionDTO && $type === DTOName::Writing ) {

            return true;

        }

        return false;

    }

public function convert($jsonObject, $options = [])

     {

        $mapper = new \JsonMapper();

        $mapper->bStrictNullTypes = false;

        $dto = $mapper->map($jsonObject, new \Test\DTOs\WritingQuestionDTO());

        return $dto;

    }

}

### Notes

Your adapter must implement ConvertToDTOAdapterInterface:

<?php

declare(strict\_types=1);

namespace Infrastructure\Convertor;

interface ConvertToDTOAdapterInterface {

    public function isHandleConvertToDTO($dtoObject, $options = []) : bool;

    public function convert($dtoObject, $options = []);

}

Convertor Service will call isHandlerConvertorToDO of each adapter. If service return true, Convertor service will call convert function of that service to convert request to DTO.

We have support basic convertor adapter ToDTOAdapter. If your data is request is sample you can extend ToDTOAdapter and return DTO class to convert

Ex:

<?php

declare(strict\_types=1);

namespace Test\Convertor\Adapter\DTOs;

use Infrastructure\Convertor\ToDTOAdapter;

use Test\Enum\DTOName;

class ToTypeDTOAdapter extends ToDTOAdapter {

    public function isHandleConvertToDTO($dtoObject, $options = []) : bool

    {

        if (isset($options[\Config\AppConstant::DTOKey]) && $options[\Config\AppConstant::DTOKey] === DTOName::TypeDTO) {

            return true;

        }

        return false;

    }

    public function getDTOClass() {

        return \Test\DTOs\Question\TypeDTO::class;

    }

}

Register your convertor

add your adapter into return of getConvertorDTOAdapters function in ConfigProvider.php at your module.

public function getConvertorDTOAdapters(): array {

        return [

            \Test\Convertor\Adapter\DTOs\ToListeningDTOAdapter::class,

            \Test\Convertor\Adapter\DTOs\ToReadingDTOAdapter::class,

            \Test\Convertor\Adapter\DTOs\ToVerbalDTOAdapter::class,

{put your convert adapter here}

        ];

    }

## Add new convert DTO to Document

In order to save object to mongodb, DTO must be convert to document, so we need to define convertor for converting.

### Add new convertor

Ex: Create adapter to convert WritingDTO object to WritingDocument.

In {Project-folder}/{your-module}/src/convertor/Documents add new file ToWritingDocumentAdapter and add code like bellow:

<?php

declare(strict\_types=1);

namespace Test\Convertor\Adapter\Documents;

use Infrastructure\Convertor\ConvertDTOAToDocumentAdapterInterface;

use Test\Services\Interfaces\SourceServiceInterface;

use Test\Services\Interfaces\TypeServiceInterface;

class ToWritingDocumentAdapter implements ConvertDTOAToDocumentAdapterInterface {

    protected $container;

    protected $convertor;

    /\*\*

     \* Class constructor.

     \*/

    public function \_\_construct($container, $convertor)

    {

        $this->container = $container;

        $this->convertor = $convertor;

    }

    public function isHandleConvertDTOToDocument($dtoObject, $options = []) : bool

    {

        if ($dtoObject instanceof \Test\DTOs\Question\WritingQuestionDTO && !isset($options[\Config\AppConstant::ToDocumentClass])) {

            return true;

        }

        return false;

    }

    public function convert($dto, $options = [])

    {

        $document = new \Test\Documents\Question\WritingQuestionDocument();

        if (isset($options[\Config\AppConstant::ExistingDocument])) {

            $document = $options[\Config\AppConstant::ExistingDocument];

        }

        $document->setContent($dto->getContent());

        $sourceService = $this->container->get(SourceServiceInterface::class);

        $sourceDocument = $sourceService->getSourceByName($dto->getSource());

        if (!$sourceDocument) {

            $translator = $this->container->get(\Config\AppConstant::Translator);

            $message = $translator->translate('Source not found, please check it again.');

            throw new \Infrastructure\Exceptions\DataException($message);

        }

        $document->setSource($sourceDocument);

        $typeService = $this->container->get(TypeServiceInterface::class);

        $typeDocument = $typeService->getTypeById($dto->getTypeId());

        if (!$typeDocument) {

            $translator = $this->container->get(\Config\AppConstant::Translator);

            $message = $translator->translate('Type not found, please check it again.');

            throw new \Infrastructure\Exceptions\DataException($message);

        }

        $document->setType($typeDocument);

        $document->setTypeId($typeDocument->getId());

        $document->setParentTypeId($typeDocument->getParentType()->getId());

        return $document;

    }

}

### Notes

Your adapter must implement ConvertDTOAToDocumentAdapterInterface:

<?php

declare(strict\_types=1);

namespace Infrastructure\Convertor;

interface ConvertDTOAToDocumentAdapterInterface {

    public function isHandleConvertDTOToDocument($object, $options = []) : bool;

    public function convert($dtoObject, $options = []);

}

Convertor Service will call isHandleConvertDTOToDocument of each adapter. If service return true, Convertor service will call convert function of that service to convert DTO to Document

Register your convertor

Add your adapter into return of getConvertorDocumentAdapters function in ConfigProvider.php at your module.

public function getConvertorDocumentAdapters (): array {

        return [

            \Test\Convertor\Adapter\Documents\ToListeningDocumentAdapter::class,

            \Test\Convertor\Adapter\Documents\ToReadingDocumentAdapter::class,

            \Test\Convertor\Adapter\Documents\ToWritingDocumentAdapter::class,

            \Test\Convertor\Adapter\Documents\ToVerbalDocumentAdapter::class,

{put your convert adapter here}

        ];

    }

### To get document object from DTO use bellow code

$convertor = $this->container->get(DTOToDocumentConvertorInterface::class);

$questionDocument = $convertor->convertToDocument($questionDTO);

## Add new convertor Document to DTO

Document object only use to work with database, when we need to communication with other service, other layer, we need to convert Document to DTO object. To convert document to object you must define adapter.

### Add new adapter

Ex: Create adapter to convert Document to DTO object.

In {Project-folder}/{your-module}/src/convertor/DTOs add new file FromWritingDocumentAdapter and add code like bellow:

<?php

declare(strict\_types=1);

namespace Test\Convertor\Adapter\DTOs;

use Infrastructure\Convertor\ConvertDocumentToDTOAdapterInterface;

use Infrastructure\Convertor\DocumentToDTOConvertorInterface;

class FromWritingDocumentAdapter implements ConvertDocumentToDTOAdapterInterface {

    private $container;

    /\*\*

     \* Class constructor.

     \*/

    public function \_\_construct($container)

    {

        $this->container = $container;

    }

    public function isHandleConvertDocumentToDTO($document, $options = []) : bool

    {

        if ($document instanceof \Test\Documents\Question\WritingQuestionDocument) {

            return true;

        }

        return false;

    }

    public function convert($document, $options = []) {

        $dto = new \Test\DTOs\Question\WritingQuestionDTO();

        $content = \Infrastructure\CommonFunction::revertToHost($document->getContent());

        $dto->setContent($content);

        $dto->setId($document->getId());

        $dto->setType($document->getType()->getParentType()->getName());

        $dto->setSubType($document->getType()->getName());

        $dto->setTypeId($document->getType()->getId());

        $dto->setSource($document->getSource()->getName());

        $dto->setSourceId($document->getSource()->getId());

        $dto->setMark($document->getMark());

        $documentToDTOConvertor = $this->container->get(DocumentToDTOConvertorInterface::class);

        return $dto;

    }

}

### Notes

Your adapter must implement ConvertDocumentToDTOAdapterInterface:

<?php

declare(strict\_types=1);

namespace Infrastructure\Convertor;

interface ConvertDocumentToDTOAdapterInterface {

    public function isHandleConvertDocumentToDTO($object, $options = []) : bool;

    public function convert($dtoObject, $options);

}

Convertor Service will call isHandleConvertDocumentToDTO of each adapter. If service return true, Convertor service will call convert function of that service to convert Document to DTO.

### Register your convertor

Add your adapter into return of getConvertorDocumentToDTOAdapters function in ConfigProvider.php at your module.

Ex:

public function getConvertorDocumentToDTOAdapters (): array {

        return [

            \Test\Convertor\Adapter\DTOs\FromListeningDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromReadingDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromWritingDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromVerbalDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromSubQuestionDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromSourceDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromTypeDocumentAdapter::class,

            \Test\Convertor\Adapter\DTOs\FromSubTypeDocumentAdapter::class

{put your convert adapter here}

        ];

    }

### To get DTO object from Document use bellow code

$convertor = $this->container->get(ConvertDocumentToDTOAdapterInterface ::class);

$questionDTO = $convertor-> convertToDTO($questionDocument);

## Add new dependence to project

## Integrate SSO login with other site.

## Deploy product

## Exception handle

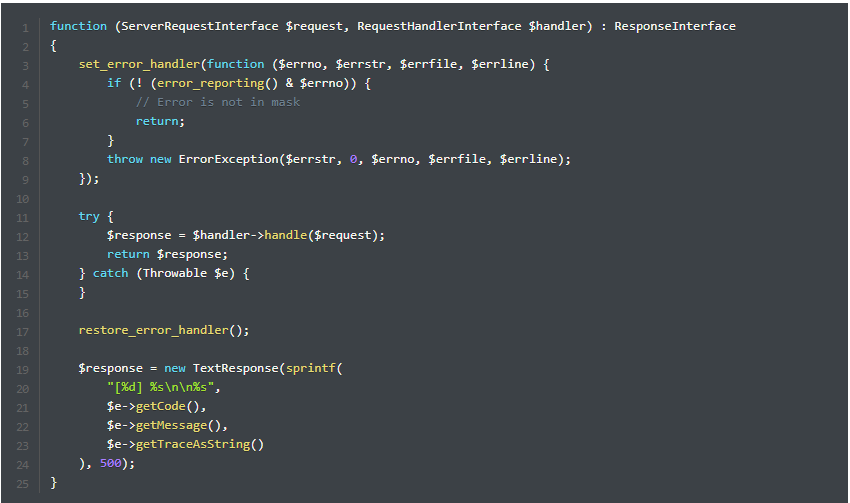
please refer link <https://docs.zendframework.com/zend-expressive/v3/features/error-handling/> to get more details

1. Steps to write middleware to handle exception.

sets an error handler that converts PHP errors to ErrorException instances.

wraps execution of the handler ($handler->handle()) with a try/catch block.

As an example:



You would then pipe this as the outermost (or close to outermost) layer of your application:



2. Using Zend middleware to handle exception.

 Zend provide an error handler, via zend tratigility: Zend\Stratigility\Middleware\ErrorHandler

Add this line to outermost in your pipe config.

$app->pipe(ErrorHandler::class);

Add these line to section factories in config dependence.

'factories' => [

ErrorResponseGenerator::class => Container\WhoopsErrorResponseGeneratorFactory::class,

'Zend\Expressive\Whoops' => Container\WhoopsFactory::class,

'Zend\Expressive\WhoopsPageHandler' => Container\WhoopsPageHandlerFactory::class,

],