CPCCore - Microframework

CPCCore is a zero configuration Micro framework built upon Zend Framework 2.0 Components.

It is ideal when you want to prototype or deliver a small project using Zend Framework 2.0 Libraries without the overhead of the ZF2 MVC. A full suite of Unit and Functional tests are included.

Usage Example...

```
<?php
require_once __DIR__ . '/../src/Bootstrap.php';

$app = new \CPCCore\Application\ApplicationService();

$app->route('root')
        ->setRoute('/')
        ->setController(function() {
            echo "Hello World";
        });

$app->run();
```

Table of Contents

Installation

Manual Installation

To install manually download from Github and install the Zend Framework 2 library into vendors under a folder called 'Zend'. You can change the location of zend by modifying the autoloader in the Bootstrap file.

Point your web server at the public folder, ensuring that mod-rewrite is enabled and your ready to go.

Production Release

The following folders are not required for production and can be removed

1. **Documentation**- User and API Guides.
2. **bin** - Build Tools

3. **tmp** - Temporary Folder for Code Coverage reports etc.
4. **tests** - Unit and functional tests.

Composer Installation

If enjoy all of the additional cruft that composer installs along with bloated autoloaders then you can install this via composer.

Create a composer file in your web folder and run. This will install all the required dependencies in the vendor folder.

```
{
    "repositories": [
        {
            "type": "vcs",
            "url": "https://github.com/CrossPlatformCoder/CPCCore.git"
        }
    ],
    "minimum-stability": "dev",
    "require": {
        "php": ">=5.5",
        "Crossplatformcoder/CPCcore": ">=1.0.0"
    }
}
```

Create an index file and your off...

```
<?php

use \CPCCore\Application\ApplicationService;

require 'vendor/autoload.php';

$app = new ApplicationService();

$app->route('root')
        ->setRoute('/')
        ->setController(function() {
            echo "Hello World";
        });

$app->run();
```

Routing

To add a route to the application simply call 'route', give the route a unique name, set the route string to match and define a controller.

```
// Basic Route
```

```
$app->route('root')
        ->setRoute('/')
        ->setController(function() {
            echo "Hello World";
        });
```

As a minimum 'setRoute' and 'setController' are required. 'setMethods' and 'setConstraints' are optional.

Route Methods

By default the following HTTP methods are matched to the route 'GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'HEAD' you can filter this by setting the methods

```
// Route will only match 'get' and 'post' methods
$app->route('root')
        ->setRoute('/')
        ->setMethods(['GET', 'POST'])
        ->setController(function() {
            echo "Hello World";
        });
```

Route Parameters

Route parameters are identified by using the character '@'. The values of the matched parameter are passed to your controller.

```
// Basic Route with parameter
$app->route('root')
        ->setRoute('/hello/@name')
        ->setController(function($name) {
            echo "Hello World";
        });
```

Route Parameter Constraints

Route parameter value matches can be constrained by using a valid regular expression.

```
// Basic Route with parameter constrained to digits
$app->route('root')
        ->setRoute('/hello/@name')
        ->setConstraints(['name' => '[0-9]+'])
        ->setController(function($name) {
            echo "Hello World";
        });
```

Controllers

When a route is matched the associated controller is called and the matched parameters are passed to it.

IMPORTANT NOTE: The parameters passed to controllers are unsanitised.

Anonymous function

```
// Basic Route with simple controller
$app->route('root')
        ->setRoute('/')
        ->setController(function($name) {
            echo "Hello World";
        });
```

Function

```
// Basic Route with simple controller
$app->route('root')
        ->setRoute('/')
        ->setController('\Namespace\MyFunction');
```

Static Method

```
// Basic Route with simple controller
$app->route('root')
        ->setRoute('/')
        ->setController('\Namespace\MyClass::StaticMethod');
```

Closure

```
// Basic Route with simple controller
$app->route('root')
        ->setRoute('/')
        ->setController(function($name) use ($app) {
            $response = $app->getResponse();
            echo "Hello World";
        });
```

Lambda

```
$lambda = function($name) {
        echo "Hello World - $name";
    });

// Basic Route with lambda controller
$app->route('root')
```

```
        ->setRoute('/')
        ->setController($lambda);
```

Class

```
class MyClass{
    function public MyMethod($name){
        echo "Hello World - $name";
    }
}

// Basic Route with class controller
$app->route('root')
        ->setRoute('/')
        ->setController(['class' => 'MyClass', 'method' => 'MyMethod']);
```

Application

Events

There are two application level events that can be hooked into 'beforeRouting' and 'afterRouting'. Any valid callbacks can be passed to these and they will be called in the order defined.

```
$app->addBeforeRouteCallback(function() {
    // Do Something here
});
$app->addAfterRouteCallback(function() {
    // Do Something here
});
$app->addAfterRouteCallback(function() {
    // Do Something here as well
});
```

Request/Response

Internally the application uses Zend Frameworks HTTP libraries for handling HTTP Requests and Responses. The can be accessed via the application object as shown below...

```
// Request :: \Zend\Http\PhpEnvironment\Request;
$request = $app->getRequest();

// Response :: \Zend\Http\PhpEnvironment\Response;
$response = $app->getResponse();
```

Registry

Values of any type and callables can be stored and retrieved within the applications registry.

Values...

```
$app->setRegistry('MyString', 'MyString-Value');
$str = $app->getRegistry('MyString');
```

Callables...

```
$lambda = function($name) {
        echo "Hello World - $name";
    });
$app->setRegistry('MyLambda', $lambda);
$obj = $app->getRegistry('MyLambda');
```

Reserved Keys

Any keys starting with 'error-' are reserved for system use. Any attempt to use these keys will cause a Registry Exception to be thrown.

```
// Will throw a Registry Exception
$app->setRegistry('error-pages', 'MyValue');
```

Error Pages

HTTP404 Page Not Found Errors

If the application cannot find a matching route it generates a HTTP404 error, returning a status code of 404 and content of "404 Error Page Not Found". This can be overridden by supplying a custom callback to 'setError404'.

```
/**
 * Response object is passed back allowing you to override any settings
 * $response is the applications response object
 * \Zend\Http\PhpEnvironment\Response;
 * Use this to ammend any error messages and codes
 */
$app->setError404(function($response) {

    // Override the preset message
    $response->setContent("My Custom 404 Message");

    // Note: Status code is already set to 404, this is for example purposes only
    $response->setStatusCode(404);
});
```

HTTP500 Server Error

Currently all other errors are returned as HTTP500 errors. In future updates I may make this more

granular by adding codes such as 501 for missing controllers.

The HTTP500 error returns a status code of 500 and renders the exeception to the ouput. This can be overridden by supplying a custom callback to 'setError500'.

```
/**
 * Response object is passed back allowing you to override any settings
 * $response is the applications response object \Zend\Http\PhpEnvironment\Response;
 * $ex is the Exception that has been thrown.
 *
 * Use this to ammend any error messages and codes
 */
$app->setError500(function($response, $ex) {

    // Override the preset message
    $response->setContent("My Custom 500 Message");

    // Note: Status code is already set to 500, this is for example purposes only
    $response->setStatusCode(500);
});
```

Helpers

Helpers are simple wrappers around ZF2 functionality.

Views

The View helper is a wrapper around the View Model and its renderer.

The view requires a path to the Templates folder passed in the constructor.

Usage....

```
$app->route('hello-world')
        ->setRoute('/hello/@name')
        ->setController(function($name) {

            // View Helper
            $view = new CPCCore\Helper\View(__DIR__ . '/../views');
            $view->setTemplate('index/hello-world.phtml');
            $view->setVariable('name', $name);
            echo $view->render();


        });
```