docs.zendframework.com

Routing and Controllers - tutorials

1. Docs »
2. MVC Tutorials »
3. Getting Started with Zend Framework »
4. Routing and Controllers

We will build a very simple inventory system to display our album collection. The home page will list our collection and allow us to add, edit and delete albums. Hence the following pages are required:

Page	Description
Home	This will display the list of albums and provide links to edit and delete them. Also, a link to enable adding new albums will be provided.
Add new album	This page will provide a form for adding a new album.
Edit album	This page will provide a form for editing an album.
Delete album	This page will confirm that we want to delete an album and then delete it.

Before we set up our files, it's important to understand how the framework expects the pages to be organised. Each page of the application is known as an *action* and actions are grouped into *controllers* within *modules*. Hence, you would generally group related actions into a controller; for instance, a news controller might have actions of `current`, `archived`, and `view`.

As we have four pages that all apply to albums, we will group them in a single controller `AlbumController` within our `Album` module as four actions. The four actions will be:

Page	Controller	Action
Home	`AlbumController`	`index`
Add new album	`AlbumController`	`add`
Edit album	`AlbumController`	`edit`
Delete album	`AlbumController`	`delete`

The mapping of a URL to a particular action is done using routes that are defined in the module's `module.config.php` file. We will add a route for our album actions. This is the updated module config file with the new code highlighted using comments.

```
 namespace Album;

use Zend\Router\Http\Segment;
use Zend\ServiceManager\Factory\InvokableFactory;

return [
    'controllers' => [
        'factories' => [
            Controller\AlbumController::class =>
InvokableFactory::class,
        ],
    ],


    'router' => [
        'routes' => [
            'album' => [
                'type'    => Segment::class,
                'options' => [
                    'route' => '/album[/:action[/:id]]',
                    'constraints' => [
                        'action' => '[a-zA-Z][a-zA-Z0-9_-]*',
                        'id'     => '[0-9]+',
                    ],
                    'defaults' => [
                        'controller' =>
Controller\AlbumController::class,
                        'action'     => 'index',
                    ],
                ],
            ],
        ],
    ],

    'view_manager' => [
        'template_path_stack' => [
            'album' => __DIR__ . '/../view',
```

```
            ],
        ],
];
```

The name of the route is 'album' and has a type of 'segment'. The segment route allows us to specify placeholders in the URL pattern (route) that will be mapped to named parameters in the matched route. In this case, the route is **`/album[/:action[/:id]]`** which will match any URL that starts with `/album`. The next segment will be an optional action name, and then finally the next segment will be mapped to an optional id. The square brackets indicate that a segment is optional. The constraints section allows us to ensure that the characters within a segment are as expected, so we have limited actions to starting with a letter and then subsequent characters only being alphanumeric, underscore, or hyphen. We also limit the id to digits.

This route allows us to have the following URLs:

URL	Page	Action
`/album`	Home (list of albums)	`index`
`/album/add`	Add new album	`add`
`/album/edit/2`	Edit album with an id of 2	`edit`
`/album/delete/4`	Delete album with an id of 4	`delete`

Create the controller

We are now ready to set up our controller. For zend-mvc, the controller is a class that is generally called `{Controller name}Controller`; note that `{Controller name}` must start with a capital letter. This class lives in a file called `{Controller name}Controller.php` within the `Controller` subdirectory for the module; in our case that is `module/Album/src/Controller/`. Each action is a public method within the controller class that is named `{action name}Action`, where `{action name}` should start with a lower case letter.

> **Conventions not strictly enforced**
>
> This is by convention. zend-mvc doesn't provide many restrictions on controllers other than that they must implement the `Zend\Stdlib\Dispatchable` interface. The framework provides two abstract classes that do this for us: `Zend\Mvc\Controller\AbstractActionController` and `Zend\Mvc\Controller\AbstractRestfulController`. We'll be using the standard `AbstractActionController`, but if you're intending to write a RESTful web service, `AbstractRestfulController` may be useful.

Let's go ahead and create our controller class in the file `zf-tutorials/module/Album/src/Controller/AlbumController.php` :

```
 namespace Album\Controller;

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

class AlbumController extends AbstractActionController
{
    public function indexAction()
    {
    }

    public function addAction()
    {
    }

    public function editAction()
    {
    }

    public function deleteAction()
    {
    }
}
```

We have now set up the four actions that we want to use. They won't work yet until we set up the views. The URLs for each action are:

URL	Method called
http://zf-tutorial.localhost/album	Album\Controller\AlbumController::indexAction
http://zf-tutorial.localhost/album/add	Album\Controller\AlbumController::addAction
http://zf-tutorial.localhost/album/edit	Album\Controller\AlbumController::editAction
http://zf-tutorial.localhost/album/delete	Album\Controller\AlbumController::deleteAction

We now have a working router and the actions are set up for each page of our application.

It's time to build the view and the model layer.

To integrate the view into our application, we need to create some view script files. These files will be executed by the `DefaultViewStrategy` and will be passed any variables or view models that are returned from the controller action method. These view scripts are stored in our module's views directory within a directory named after the controller. Create these four empty files now:

- `module/Album/view/album/album/index.phtml`
- `module/Album/view/album/album/add.phtml`
- `module/Album/view/album/album/edit.phtml`
- `module/Album/view/album/album/delete.phtml`

We can now start filling everything in, starting with our database and models.
