REPUBLIC OF THE PHILIPPINES

DEPARTMENT OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

RUFINO JOHN E. AGUILAR

https://rufdev.github.io/

Module 4 : RESTful APIs

- Building RESTful APIs using CodeIgniter.
- Authentication and security for APIs.

Building RESTful APIs

- Representational State Transfer (REST) is an architectural style for distributed applications, first described by Roy Fielding in his 2000 PhD dissertation, Architectural Styles and the Design of Network-based Software Architectures. That might be a bit of a dry read, and you might find Martin Fowler's Richardson Maturity Model a gentler introduction.

- REST has been interpreted, and mis-interpreted, in more ways than most software architectures, and it might be easier to say that the more of Roy Fielding's principles that you embrace in an architecture, the most "RESTful" your application would be considered.

Building RESTful APIs

Resource Routes

You can quickly create a handful of RESTful routes for a single resource with the `resource()` method. This creates the five most common routes needed for full CRUD of a resource: create a new resource, update an existing one, list all of that resource, show a single resource, and delete a single resource. The first parameter is the resource name:

```php
<?php

$routes->resource('photos');

// Equivalent to the following:
$routes->get('photos/new', 'Photos::new');
$routes->post('photos', 'Photos::create');
$routes->get('photos', 'Photos::index');
$routes->get('photos/(:segment)', 'Photos::show/$1');
$routes->get('photos/(:segment)/edit', 'Photos::edit/$1');
$routes->put('photos/(:segment)', 'Photos::update/$1');
$routes->patch('photos/(:segment)', 'Photos::update/$1');
$routes->delete('photos/(:segment)', 'Photos::delete/$1');
```

Building RESTful APIs

Change the Controller Used

You can specify the controller that should be used by passing in the `controller` option with the name of the controller that should be used:

```php
<?php

$routes->resource('photos', ['controller' => 'App\Gallery']);

// Would create routes like:
$routes->get('photos', 'App\Gallery::index');
```

Building RESTful APIs

Limit the Routes Made

You can restrict the routes generated with the `only` option. This should be **an array** or **comma separated list** of method names that should be created. Only routes that match one of these methods will be created. The rest will be ignored:

```php
<?php

$routes->resource('photos', ['only' => ['index', 'show']]);
```

Otherwise you can remove unused routes with the `except` option. This should also be **an array** or **comma separated list** of method names. This option run after `only`:

```php
<?php

$routes->resource('photos', ['except' => 'new,edit']);
```

Valid methods are: `index`, `show`, `create`, `update`, `new`, `edit` and `delete`.

Building RESTful APIs

ResourceController

The `ResourceController` provides a convenient starting point for your RESTful API, with methods that correspond to the resource routes above.

Extend it, over-riding the `modelName` and `format` properties, and then implement those methods that you want handled:

```php
<?php

namespace App\Controllers;

use CodeIgniter\RESTful\ResourceController;

class Photos extends ResourceController
{
    protected $modelName = 'App\Models\Photos';
    protected $format    = 'json';

    public function index()
    {
        return $this->respond($this->model->findAll());
    }

    // ...
}
```

Building RESTful APIs

Presenter Routes

You can quickly create a presentation controller which aligns with a resource controller, using the `presenter()` method. This creates routes for the controller methods that would return views for your resource, or process forms submitted from those views.

It is not needed, since the presentation can be handled with a conventional controller - it is a convenience. Its usage is similar to the resource routing:

```php
<?php

$routes->presenter('photos');

// Equivalent to the following:
$routes->get('photos/new', 'Photos::new');
$routes->post('photos/create', 'Photos::create');
$routes->post('photos', 'Photos::create');   // alias
$routes->get('photos', 'Photos::index');
$routes->get('photos/show/(:segment)', 'Photos::show/$1');
$routes->get('photos/(:segment)', 'Photos::show/$1');  // alias
$routes->get('photos/edit/(:segment)', 'Photos::edit/$1');
$routes->post('photos/update/(:segment)', 'Photos::update/$1');
$routes->get('photos/remove/(:segment)', 'Photos::remove/$1');
$routes->post('photos/delete/(:segment)', 'Photos::delete/$1');
```

Building RESTful APIs

Change the Controller Used

You can specify the controller that should be used by passing in the `controller` option with the name of the controller that should be used:

```php
<?php

$routes->presenter('photos', ['controller' => 'App\Gallery']);

// Would create routes like:
$routes->get('photos', 'App\Gallery::index');
```

Building RESTful APIs

Limit the Routes Made

You can restrict the routes generated with the `only` option. This should be **an array** or **comma separated list** of method names that should be created. Only routes that match one of these methods will be created. The rest will be ignored:

```php
<?php

$routes->presenter('photos', ['only' => ['index', 'show']]);
```

Otherwise you can remove unused routes with the `except` option. This should also be **an array** or **comma separated list** of method names. This option run after `only`:

```php
<?php

$routes->presenter('photos', ['except' => 'new,edit']);
```

Valid methods are: `index`, `show`, `new`, `create`, `edit`, `update`, `remove` and `delete`.

Building RESTful APIs

ResourcePresenter

The `ResourcePresenter` provides a convenient starting point for presenting views of your resource, and processing data from forms in those views, with methods that align to the resource routes above.

Extend it, over-riding the `modelName` property, and then implement those methods that you want handled:

```php
<?php

namespace App\Controllers;

use CodeIgniter\RESTful\ResourcePresenter;

class Photos extends ResourcePresenter
{
    protected $modelName = 'App\Models\Photos';

    public function index()
    {
        return view('templates/list', $this->model->findAll());
    }

    // ...
}
```

Authentication and security for APIs

- Controller Filters allow you to perform actions either before or after the controllers execute. Unlike events, you can choose the specific URIs or routes in which the filters will be applied to. Before filters may modify the Request while after filters can act on and even modify the Response, allowing for a lot of flexibility and power.
  - Performing CSRF protection on the incoming requests
  - Restricting areas of your site based upon their Role
  - Perform rate limiting on certain endpoints
  - Display a “Down for Maintenance” page
  - Perform automatic content negotiation

Authentication and security for APIs

The Controller Filters you can use to protect your routes Shield provides are:

```php
public $aliases = [
    // ...
    'session'      => \CodeIgniter\Shield\Filters\SessionAuth::class,
    'tokens'       => \CodeIgniter\Shield\Filters\TokenAuth::class,
    'hmac'         => \CodeIgniter\Shield\Filters\HmacAuth::class,
    'chain'        => \CodeIgniter\Shield\Filters\ChainAuth::class,
    'auth-rates'   => \CodeIgniter\Shield\Filters\AuthRates::class,
    'group'        => \CodeIgniter\Shield\Filters\GroupFilter::class,
    'permission'   => \CodeIgniter\Shield\Filters\PermissionFilter::class,
    'force-reset' => \CodeIgniter\Shield\Filters\ForcePasswordResetFilter::class,
    'jwt'          => \CodeIgniter\Shield\Filters\JWTAuth::class,
];
```

Authentication and security for APIs

Filters	Description
session	The `Session` authenticator.
tokens	The `AccessTokens` authenticator.
chained	The filter will check authenticators in sequence to see if the user is logged in through either of authenticators, allowing a single API endpoint to work for both an SPA using session auth, and a mobile app using access tokens.
jwt	The `JWT` authenticator. See JWT Authentication.
hmac	The `HMAC` authenticator. See HMAC Authentication.
auth-rates	Provides a good basis for rate limiting of auth-related routes.
group	Checks if the user is in one of the groups passed in.
permission	Checks if the user has the passed permissions.
force-reset	Checks if the user requires a password reset.

Authentication and security for APIs

Configure Controller Filters

Protect All Pages

If you want to limit all routes (e.g. `localhost:8080/admin`, `localhost:8080/panel` and ...), you need to add the following code in the **app/Config/Filters.php** file.

```
public $globals = [
    'before' => [
        // ...
        'session' => ['except' => ['login*', 'register', 'auth/a/*']],
    ],
    // ...
];
```

Authentication and security for APIs

Rate Limiting

To help protect your authentication forms from being spammed by bots, it is recommended that you use the `auth-rates` filter on all of your authentication routes. This can be done with the following filter setup:

```
public $filters = [
    'auth-rates' => [
        'before' => [
            'login*', 'register', 'auth/*'
        ]
    ]
];
```

Authentication and security for APIs

Forcing Password Reset

If your application requires a force password reset functionality, ensure that you exclude the auth pages and the actual password reset page from the `before` global. This will ensure that your users do not run into a *too many redirects* error. See:

```
public $globals = [
    'before' => [
        //...
        //...
        'force-reset' => ['except' => ['login*', 'register', 'auth/a/*', 'change-pa
    ]
];
```

Authentication and security for APIs

Auth Helper

The auth functionality is designed to be used with the `auth_helper` that comes with Shield. This helper method provides the `auth()` function which returns a convenient interface to the most frequently used functionality within the auth libraries.

```
// get the current user
auth()->user();

// get the current user's id
auth()->id();
// or
user_id();

// get the User Provider (UserModel by default)
auth()->getProvider();
```

Authentication and security for APIs

Authorizing Users

The `Authorizable` trait on the `User` entity provides the following methods to authorize your users.

can()

Allows you to check if a user is permitted to do a specific action or group or actions. The permission string(s) should be passed as the argument(s). Returns boolean `true` / `false`. Will check the user's direct permissions (**user-level permissions**) first, and then check against all of the user's groups permissions (**group-level permissions**) to determine if they are allowed.

```
if ($user->can('users.create')) {
    //
}

// If multiple permissions are specified, true is returned if the user has any of t
if ($user->can('users.create', 'users.edit')) {
    //
}
```

Authentication and security for APIs

inGroup()

Checks if the user is in one of the groups passed in. Returns boolean `true` / `false`.

```
if (! $user->inGroup('superadmin', 'admin')) {
    //
}
```

Authentication and security for APIs

hasPermission()

Checks to see if the user has the permission set directly on themselves. This disregards any groups they are part of.

```php
if (! $user->hasPermission('users.create')) {
    //
}
```

Authentication and security for APIs

Authorizing via Routes

You can restrict access to a route or route group through a Controller Filter.

One is provided for restricting via groups the user belongs to, the other is for permission they need. The filters are automatically registered with the system under the `group` and `permission` aliases, respectively.

You can set the filters within **app/Config/Routes.php**:

```
$routes->group('admin', ['filter' => 'group:admin,superadmin'], static function ($r
    $routes->group(
        '',
        ['filter' => ['group:admin,superadmin', 'permission:users.manage']],
        static function ($routes) {
            $routes->resource('users');
        }
    );
});
```

REPUBLIC OF THE PHILIPPINES
DEPARTMENT OF INFORMATION AND COMMUNICATIONS TECHNOLOGY