chriskacerguis / **codeigniter-restserver**

A fully RESTful server implementation for CodeIgniter using one library, one config file and one controller.

843 commits | **1** branch | **18** releases | **122** contributors | MIT

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

chriskacerguis remove unneeded badge — Latest commit `9791010` on Nov 25, 2017

application	special chars encoding (#858)	4 months ago
documentation	Re-generated documentation	2 years ago
.codeclimate.yml	exclude languages	2 years ago
.gitignore	Code smell, PHP Doc problems, Index methods, and global exception han...	a year ago
AUTHORS.md	fix typo	4 months ago
CHANGELOG.md	All around cleaning (#734)	a year ago
LICENSE	Removed .txt extension from LICENSE	2 years ago
README.md	remove unneeded badge	2 months ago
composer.json	Codeigniter Composer Ready (#832)	5 months ago

README.md

CodeIgniter Rest Server

A fully RESTful server implementation for CodeIgniter using one library, one config file and one controller.

Requirements

1. PHP 5.4 or greater
2. CodeIgniter 3.0+

Note: for 1.7.x support download v2.2 from Downloads tab

Important Update on 4.0.0

Please note that version 4.0.0 is in the works, and is considered a breaking change (per SemVer). As CI 3.1.0 now has native support for Composer, this library will be moving to be composer based.

Take a look at the "development" branch to see what's up.

Installation & loading

CodeIgniter Rest Server is available on Packagist (using semantic versioning), and installation via composer is the recommended way to install Codeigniter Rest Server. Just add this line to your `composer.json` file:

```
"chriskacerguis/codeigniter-restserver": "^3.0"
```

or run

```
composer require chriskacerguis/codeigniter-restserver
```

Handling Requests

When your controller extends from `REST_Controller`, the method names will be appended with the HTTP method used to access the request. If you're making an HTTP `GET` call to `/books`, for instance, it would call a `Books#index_get()` method.

This allows you to implement a RESTful interface easily:

```
class Books extends REST_Controller
{
  public function index_get()
  {
    // Display all books
  }

  public function index_post()
  {
    // Create a new book
  }
}
```

`REST_Controller` also supports `PUT` and `DELETE` methods, allowing you to support a truly RESTful interface.

Accessing parameters is also easy. Simply use the name of the HTTP verb as a method:

```
$this->get('blah'); // GET param
$this->post('blah'); // POST param
$this->put('blah'); // PUT param
```

The HTTP spec for DELETE requests precludes the use of parameters. For delete requests, you can add items to the URL

```
public function index_delete($id)
{
        $this->response([
                'returned from delete:' => $id,
        ]);
}
```

If query parameters are passed via the URL, regardless of whether it's a GET request, can be obtained by the query method:

```
$this->query('blah'); // Query param
```

Content Types

`REST_Controller` supports a bunch of different request/response formats, including XML, JSON and serialised PHP. By default, the class will check the URL and look for a format either as an extension or as a separate segment.

This means your URLs can look like this:

```
http://example.com/books.json
http://example.com/books?format=json
```

This can be flaky with URI segments, so the recommend approach is using the HTTP `Accept` header:

```
$ curl -H "Accept: application/json" http://example.com
```

Any responses you make from the class (see responses for more on this) will be serialised in the designated format.

Responses

The class provides a `response()` method that allows you to return data in the user's requested response format.

Returning any object / array / string / whatever is easy:

```
public function index_get()
{
  $this->response($this->db->get('books')->result());
}
```

This will automatically return an `HTTP 200 OK` response. You can specify the status code in the second parameter:

```
public function index_post()
  {
    // ...create new book
    $this->response($book, 201); // Send an HTTP 201 Created
  }
```

If you don't specify a response code, and the data you respond with `== FALSE` (an empty array or string, for instance), the response code will automatically be set to `404 Not Found`:

```
$this->response([]); // HTTP 404 Not Found
```

Configuration

You can overwrite all default configurations by creating a rest.php file in your config folder with your configs. All given configurations will overwrite the default ones.

Language

You can overwrite all default language files. Just add a rest_controller_lang.php to your language and overwrite the what you want.

Multilingual Support

If your application uses language files to support multiple locales, `REST_Controller` will automatically parse the HTTP `Accept-Language` header and provide the language(s) in your actions. This information can be found in the `$this->response->lang` object:

```
public function __construct()
{
  parent::__construct();

  if (is_array($this->response->lang))
  {
    $this->load->language('application', $this->response->lang[0]);
  }
  else
  {
    $this->load->language('application', $this->response->lang);
  }
}
```

Authentication

This class also provides rudimentary support for HTTP basic authentication and/or the securer HTTP digest access authentication.

You can enable basic authentication by setting the `$config['rest_auth']` to `'basic'`. The `$config['rest_valid_logins']` directive can then be used to set the usernames and passwords able to log in to your system. The class will automatically send all the correct headers to trigger the authentication dialogue:

```
$config['rest_valid_logins'] = ['username' => 'password', 'other_person' => 'secure123'];
```

Enabling digest auth is similarly easy. Configure your desired logins in the config file like above, and set `$config['rest_auth']` to `'digest'`. The class will automatically send out the headers to enable digest auth.

If you're tying this library into an AJAX endpoint where clients authenticate using PHP sessions then you may not like either of the digest nor basic authentication methods. In that case, you can tell the REST Library what PHP session variable to check for. If the variable exists, then the user is authorized. It will be up to your application to set that variable. You can define the variable in `$config['auth_source']`. Then tell the library to use a php session variable by setting `$config['rest_auth']` to `session`.

All three methods of authentication can be secured further by using an IP white-list. If you enable `$config['rest_ip_whitelist_enabled']` in your config file, you can then set a list of allowed IPs.

Any client connecting to your API will be checked against the white-listed IP array. If they're on the list, they'll be allowed access. If not, sorry, no can do hombre. The whitelist is a comma-separated string:

```
$config['rest_ip_whitelist'] = '123.456.789.0, 987.654.32.1';
```

Your localhost IPs (`127.0.0.1` and `0.0.0.0`) are allowed by default.

API Keys

In addition to the authentication methods above, the `REST_Controller` class also supports the use of API keys. Enabling API keys is easy. Turn it on in your **config/rest.php** file:

```
$config['rest_enable_keys'] = TRUE;
```

You'll need to create a new database table to store and access the keys. `REST_Controller` will automatically assume you have a table that looks like this:

```
CREATE TABLE `keys` (
        `id` INT(11) NOT NULL AUTO_INCREMENT,
        `key` VARCHAR(40) NOT NULL,
        `level` INT(2) NOT NULL,
        `ignore_limits` TINYINT(1) NOT NULL DEFAULT '0',
        `date_created` INT(11) NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The class will look for an HTTP header with the API key on each request. An invalid or missing API key will result in an `HTTP 403 Forbidden`.

By default, the HTTP will be `X-API-KEY`. This can be configured in **config/rest.php**.

```
$ curl -X POST -H "X-API-KEY: some_key_here" http://example.com/books
```

Profiling

Codeigniter Profiler feature has been added to the library, so that you can use the power of CI profiler in your project just by setting config parameter to enable profile through out your application Turn it on in your **config/config.php** file:

```
TRUE to turn profile ON, FALSE to turn it off
$config['enable_profiling'] = FALSE;
```

Also you need to enable `hooks` in your config.php that looks like this

```
$config['enable_hooks'] = TRUE;
```

Also you can refer to **config/config.php.sample**

Other Documentation / Tutorials

- NetTuts: Working with RESTful Services in CodeIgniter

Contributions

This project was originally written by Phil Sturgeon, however his involvement has shifted as he is no longer using it. As of 2013/11/20 further development and support will be done by Chris Kacerguis.

Pull Requests are the best way to fix bugs or add features. I know loads of you use this, so please contribute if you have improvements to be made and I'll keep releasing versions over time.