laravel-restive Documentation

Release 1.0

Ian Wilson

Aug 28, 2020

Contents

Restive is written for Laravel and provides a query parser and api controllers for CRUD type actions.

The query parser allows for complex filtering and sorting, converting the URI query into eloquent queries.

The API controller supports resource creation, reading, updating and deletion.

Reading, updating and deletion can all access the query parser.

e.g.

```
whereIn
whereBetween
sort
```

although note that the filtering can be more complex than just a simple `where`

CHAPTER 1

Installation

Installation is via composer.

```
composer require laravel-restive/restive
```

CHAPTER 2

Query Parser

The query parser allows for complex filtering, sorting, the use of child relations and more.

Currently the filter parser supports

- with
- where
- orWhere
- whereIn
- orWhereIn
- whereNotIn
- orWhereNotIn
- whereBetween
- whereNotBetween
- orWhereBetween
- orWhereNotBetween
- withTrashed
- onlyTrashed
- scope

Sorting allows for multiple sort targets for ascending and descending sorts.

Includes allow for loading child models.

Joins are also supported

Query results by default return all columns for the query, however you can use the columns filter to restrict which columns are returned.

2.1 Query Parser Filtering

2.1.1 Simple Where Clauses

Simple where clauses take the format of

```
where[]=fieldname:operator:value
```

for example

```
?where[]=id:eq:1
```

would equate to an eloquent query of

```
model::where('id', '=', 1)
```

The operators allowed are

- eq equates to =
- noteq equates to !=
- lte equates to <=
- gte equates to >=
- gt equates to >
- lt equates to <
- lk equates to LIKE
- nlk equates to NOT LIKE

Can also use orWhere

```
orWhere[]=id:eq:1
```

2.1.2 Where In clauses

Where In clauses take the form of

```
whereIn[]=fieldname:(comma separate list)
```

For example

```
whereIn[]=id:(1,2,3)
```

Can also use

- orWhereIn
- whereNotIn
- orWhereNotIn

2.1.3 Where Between Clauses

Where Between clauses take the form of

```
whereBetween[]=fieldname:start:end
```

For example

```
whereBetween[]=age:18:45
```

Can also use

- orWhereBetween
- whereNotBetween
- orWhereNotBetween

2.2 Query Parser Soft Deletes

For models that support soft deletes, the query parser also provides 2 further filters.

2.2.1 With Trashed

To show all entries for the model, even when soft deleted use

```
withTrashed
```

2.2.2 Only Trashed

To show only entries for the model that have been soft deleted use

```
onlyTrashed
```

Note: Using `withTrashed` or `onlyTrashed` on models that do not support `soft deletes` will result in an exception, with the error message being returned in the Json response.

e.g.

```
{"error":{"message":"Model does not support soft deletes","status_code":400}}
```

Warning: There is currently no way to force delete soft deleted items. This is on the todo list.

2.3 Query Parser Sorting

example

```
sort[]=id,-name
```

would sort ascending on id then sort descending on name

2.4 Query Parser Columns

example

```
columns[]=id,name
```

will restrict the returned columns to just id and name.

2.5 Query Parser With

Warning: This feature is a bit experimental at the moment. In terms of testing i've only tried with a simple one to many relationship. e.g. `user->posts`.

example

```
with[]=posts
```

The above assumes the query is being done on a model that has a relationship defined, and uses the Laravel Querbuilder `with` method.

2.6 Query Parser Joins

Example

```
join[]=joinType:tableName:leftKey:rightKey
```

The join clause takes 4 parameters

- joinType - can be 1 of `inner`, `left` or `cross`
- tableName - the table to join on
- leftkey - the table field used on the left side of the Join on clause
- rightKey - the table field used on the right side of the Join on clause

More examples

```
join[]=inner:posts:posts.user_id:users.id

is the same as

$model->join('posts', 'posts.user_id', '=', 'users.id', 'inner');
```

2.7 Query Parser Scopes

Warning: This feature is a bit experimental at the moment.

example

```
scope[]=myscope
```

Allows the use of Laravel scopes

the *scope* parameter should be the name of your scope less the preceding *scope*

e.g. If your scope is called *scopeActive* then you would just use *active*

2.8 URL Parameter Format

For `GET` e.g. `index` routes then the parser parameters can be placed in the url.

for example.

```
{api-uri}?columns[]=id,name&where[]=id:eq:1&orWhereBetween[]=age:(10,
15)&orWhereBetween[]=age:(50,60)
```

For `DELETE` and `PUT` the parser parameters are added to the body of the normal request, inside a parameter called `@parser`

For example

```
$response = $this->put("/user", [
    'email' => 'dirk2@holisticdetective.com',
    'name' => 'Dirk Gently',
    'age' => 45,
    '@parser' => ['where' => ['email:eq:dirk@holisticdetective.com']]
]);
```

CHAPTER 3

API Controllers

3.1 Routes

zcwilt/rest-api uses resource controllers. To define your routes for each controller you want, you will need to add the following to your api routes file.

```
Restive::resource('modelName', 'controller');
```

`modelName` is the model name you want to have api access.

`Controller` is the name of your controller class.

As an example. To use the dummy simple controller supplied by the project, your routing entries would be.

```
Restive::resource('dummySimple', 'Api\DummySimpleController');
```

or to use the default User model that comes with Laravel

```
Restive::resource('user', 'Api\UserController');
```

For each Laravel Model that you want to use in the API you will need to create a Controller

3.2 Controller Definition

As mentioned above, for each Laravel model that you want to provide API access to you will need to create a Controller.

This should be placed in the standard laravel location e.g. App/Http/Controllers or a sub directory. Our suggestion is to use App/Http/Controllers/Api

The controller definition is fairly simple

```
<?php
namespace App\Http\Controllers\Api;

use Restive\Controllers\ApiController;

class DummySimpleController extends ApiController
{
    protected $modelName = '\\Restive\\Models\\DummySimple';
}
```

Note: The `protected $modelName` defines the Eloquent Model that will be used by the controller. The factory class used will try and resolve the model from either your projects App folder or from the App/Models folder, If the Model is in one of these folders there is no need to namespace the model name. e.g. you could just do

```
protected $modelName = 'ModelName';
```

3.3 Validation

The base ApiController class that your controller extends contains calls to Laravels Validation system.

To use validation on your api request you must create a public `rules` method on the model your controller accesses.

e.g.

```
public function rules($id = 0)
{
    return [
        'email' => 'required|unique:zcwilt_users'.($id ? ",email,$id" : ''),
        'name' => 'required'
    ];
}
```

Note: For update methods, the primary key value is passed as the `$id` parameter.

Warning: The controller uses `$request->all()` to pass request fields to the model update/create methods. This could allow malicious users to update database fields that you did not want. You **must** therefore be very explicit in your models as to which fields can be used in this way. e.g. using the `$fillable` property.

3.4 Api Endpoints

The api endpoints provided by the resource controller and extra controller methods provide the following route/actions

```
GET api/modelname -> controller@index : allows for query filtering on the url
```

```
GET api/modelname/{id} -> controller@show
```

```
POST api/modelname -> controller@store
The request body should be an array of field/values
e.g ['name' => 'foo', 'email' => 'bar@test.com']
```

```
PUT api/modelname/{id} -> controller@update
The request body should be an array of field/values
e.g ['name' => 'foo', 'email' => 'bar@test.com']

Updates can use filtering e.g.
PUT api/modelname -> controller@update
while including something like
['@parser' => ['whereBetween' => ['age:21:65]]
```

```
DELETE api/modelname/{id} - controller@destroy

Deletes can use filtering e.g.
DELETE api/modelname -> controller@destroy
while including something like
['@parser' => ['whereBetween' => ['age:21:65]]
```

3.5 Pagination

All results from the `index` route are paginated using the standard Laravel paginator Therefore you can add a `page` and `per_page` parameter to those queries. You can also return all results by adding `paginate=no` to the query string.

CHAPTER 4

Authorization

This project/code is agnostic as to how you provide authorization for your API.

It's expected that you may need to extend the Api Controller to provide authorization and/or role/scope based access.

There will be some more documentation here regarding Authorizartion etc

CHAPTER 5

Testing

Tests can be run within the package

```
``vendor/bin/phpunit``

or

``composer tests``
```

There is also a phpunit xml file for running tests with code coverage

```
``vendor/bin/phpunit -c phpunit-cc.xml``

or

``composer coverage``
```

Note: To run the code coverage tests, you will need to install xdebug

CHAPTER 6

Contributing

See the - Contributing File

CHAPTER 7

Acknowledgements

This project would not exist without the work of others.

Thanks go to

- Laravel Framework
- Read The Docs and Sphinx

From our composer.json

- orchestra/testbench
- phpunit
- codedungeon/phpunit-result-printer
- nunomaduro/collision

The following projects also provided inspiration.

- spatie/laravel-query-builder
- esbenp/bruno
- dingo/api
- Froiden/laravel-rest-api

Thanks also to the guys from Reddit who provided feedback

- /u/tusconflyer
- /u/somethingeneric

CHAPTER 8

License

This code is released under the MIT license.

See the License File

CHAPTER 9

Todo

- Allow for force deleting entries
- Document Exception Handling
- Update demo site with auth examples