rakit / **validation**

PHP Standalone Validation Library

#validation #php #php-library #php-standalone-library

109 commits | **3** branches | **20** releases | **3** contributors

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

emsifa Merge pull request #25 from Schoolrunner/feature/humanized_attribute_key … Latest commit 40837c4 17 days ago

src	Allow turning humanized keys on/off.	19 days ago
tests	Changed test to work with lower versions of php	4 months ago
.gitattributes	Add .gitattribute to exclude tests files	a year ago
.gitignore	initialize commit	a year ago
.travis.yml	initialize commit	a year ago
README.md	Add documentation for callback rule	5 months ago
composer.json	typo requirement php version	a year ago
phpunit.xml.dist	Adding code covergage filter	a year ago

README.md

Rakit Validation - PHP Standalone Validation Library

build passing | license mit

PHP Standalone library for validating data. Inspired by `Illuminate\Validation` Laravel.

Requirements

- PHP 5.5 or higher
- Composer for installation

Quick Start

Installation

```
composer require "rakit/validation"
```

Usage

There are two ways to validating data with this library. Using `make` to make validation object, then validate it using `validate`. Or just use `validate`. Examples:

Using `make`:

```
<?php

require('vendor/autoload.php');

use Rakit\Validation\Validator;

$validator = new Validator;

// make it
$validation = $validator->make($_POST + $_FILES, [
    'name'                  => 'required',
    'email'                 => 'required|email',
    'password'              => 'required|min:6',
    'confirm_password'      => 'required|same:password',
    'avatar'                => 'required|uploaded_file:0,500K,png,jpeg',
    'skills'                => 'array',
    'skills.*.id'           => 'required|numeric',
    'skills.*.percentage'   => 'required|numeric'
]);

// then validate
$validation->validate();

if ($validation->fails()) {
    // handling errors
    $errors = $validation->errors();
    echo "<pre>";
    print_r($errors->firstOfAll());
    echo "</pre>";
    exit;
} else {
    // validation passes
    echo "Success!";
}
```

or just `validate` it:

```
<?php

require('vendor/autoload.php');

use Rakit\Validation\Validator;

$validator = new Validator;

$validation = $validator->validate($_POST + $_FILES, [
    'name'                  => 'required',
    'email'                 => 'required|email',
    'password'              => 'required|min:6',
    'confirm_password'      => 'required|same:password',
    'avatar'                => 'required|uploaded_file:0,500K,png,jpeg',
    'skills'                => 'array',
    'skills.*.id'           => 'required|numeric',
    'skills.*.percentage'   => 'required|numeric'
]);

if ($validation->fails()) {
        // handling errors
        $errors = $validation->errors();
        echo "<pre>";
        print_r($errors->firstOfAll());
        echo "</pre>";
        exit;
} else {
        // validation passes
        echo "Success!";
}
```

In this case, 2 examples above will output the same results.

But with `make` you can setup something like custom invalid message, custom attribute alias, etc before validation running.

Attribute Alias

By default we will transform your attribute into more readable text. For example `confirm_password` will be displayed as `Confirm password`. But you can set it anything you want with `setAlias` or `setAliases` method.

Example:

```
$validator = new Validator;

// To set attribute alias, you should use `make` instead `validate`.
$validation->make([
        'province_id' => $_POST['province_id'],
        'district_id' => $_POST['district_id']
], [
        'province_id' => 'required|numeric',
        'district_id' => 'required|numeric'
]);

// now you can set aliases using this way:
$validation->setAlias('province_id', 'Province');
$validation->setAlias('district_id', 'District');

// or this way:
$validation->setAliases([
        'province_id' => 'Province',
        'district_id' => 'District'
]);

// then validate it
$validation->validate();
```

Now if `province_id` value is empty, error message would be 'Province is required'.

Custom Validation Message

Before register/set custom messages, here are some variables you can use in your custom messages:

- `:attribute`: will replaced into attribute alias.
- `:value`: will replaced into stringify value of attribute. For array and object will replaced to json.

And also there are several message variables depends on their rules.

Here are some ways to register/set your custom message(s):

Custom Messages for Validator

With this way, anytime you make validation using `make` or `validate` it will set your custom messages for it. It is useful for localization.

To do this, you can set custom messages as first argument constructor like this:

```
$validator = new Validator([
        'required' => ':attribute harus diisi',
        'email' => ':email tidak valid',
        // etc
]);

// then validation belows will use those custom messages
$validation_a = $validator->validate($dataset_a, $rules_for_a);
$validation_b = $validator->validate($dataset_b, $rules_for_b);
```

Or using `setMessages` method like this:

```
$validator = new Validator;
$validator->setMessages([
        'required' => ':attribute harus diisi',
        'email' => ':email tidak valid',
        // etc
]);

// now validation belows will use those custom messages
$validation_a = $validator->validate($dataset_a, $rules_for_dataset_a);
$validation_b = $validator->validate($dataset_b, $rules_for_dataset_b);
```

Custom Messages for Validation

Sometimes you may want to set custom messages for specific validation. To do this you can set your custom messages as 3rd argument of `$validator->make` or `$validator->validate` like this:

```
$validator = new Validator;
```

```
$validation_a = $validator->validate($dataset_a, $rules_for_dataset_a, [
        'required' => ':attribute harus diisi',
        'email' => ':email tidak valid',
        // etc
]);
```

Or you can use `$validation->setMessages` like this:

```
$validator = new Validator;

$validation_a = $validator->make($dataset_a, $rules_for_dataset_a);
$validation_a->setMessages([
        'required' => ':attribute harus diisi',
        'email' => ':email tidak valid',
        // etc
]);

...

$validation_a->validate();
```

Custom Message for Specific Attribute Rule

Sometimes you may want to set custom message for specific rule attribute. To do this you can use `:` as message separator or using chaining methods.

Examples:

```
$validator = new Validator;

$validation_a = $validator->make($dataset_a, [
        'age' => 'required|min:18'
]);

$validation_a->setMessages([
        'age:min' => '18+ only',
]);

$validation_a->validate();
```

Or using chaining methods:

```
$validator = new Validator;

$validation_a = $validator->make($dataset_a, [
        'photo' => [
                'required',
                $validator('uploaded_file')->fileTypes('jpeg|png')->message('Photo must be jpeg/png image')
        ]
]);

$validation_a->validate();
```

Available Rules

Below is list of all available validation rules

- required
- required_if
- required_unless
- required_with
- required_without
- required_with_all
- required_without_all
- uploaded_file
- email

- alpha
- numeric
- alpha_num
- alpha_dash
- in
- not_in
- min
- max
- between
- url
- ip
- ipv4
- ipv6
- array
- same
- regex
- date
- accepted
- present
- different
- after
- before
- callback

required

The field under this validation must be present and not 'empty'.

Here are some examples:

Value	Valid
`'something'`	true
`'0'`	true
`0`	true
`[0]`	true
`[null]`	true
null	false
[]	false
''	false

For uploaded file, `$_FILES['key']['error']` must not `UPLOAD_ERR_NO_FILE`.

required_if:another_field,value_1,value_2,...

The field under this rule must be present and not empty if the anotherfield field is equal to any value.

For example `required_if:something,1,yes,on` will be required if `something` value is one of `1`, `'1'`, `'yes'`, or `'on'`.

required_unless:another_field,value_1,value_2,...

The field under validation must be present and not empty unless the anotherfield field is equal to any value.

required_with:field_1,field_2,...

The field under validation must be present and not empty only if any of the other specified fields are present.

required_without:field_1,field_2,...

The field under validation must be present and not empty only when any of the other specified fields are not present.

required_with_all:field_1,field_2,...

The field under validation must be present and not empty only if all of the other specified fields are present.

required_without_all:field_1,field_2,...

The field under validation must be present and not empty only when all of the other specified fields are not present.

uploaded_file:min_size,max_size,file_type_a,file_type_b,...

This rule will validate `$_FILES` data, but not for multiple uploaded files. Field under this rule must be following rules below to be valid:

- `$_FILES['key']['error']` must be `UPLOAD_ERR_OK` or `UPLOAD_ERR_NO_FILE`. For `UPLOAD_ERR_NO_FILE` you can validate it with `required` rule.
- If min size is given, uploaded file size **MUST NOT** be lower than min size.
- If max size is given, uploaded file size **MUST NOT** be higher than max size.
- If file types is given, mime type must be one of those given types.

Here are some example definitions and explanations:

Definition	Explanation
`uploaded_file`	Uploaded file is optional. When it is not empty, it must be `ERR_UPLOAD_OK`.
`required	uploaded_file`
`uploaded_file:0,1M`	uploaded file size must be between 0 - 1 MB, but uploaded file are optional
`required	uploaded_file:0,1M,png,jpeg`

email

The field under this validation must be valid email address.

alpha

The field under this rule must be entirely alphabetic characters.

numeric

The field under this rule must be numeric.

alpha_num

The field under this rule must be entirely alpha-numeric characters.

alpha_dash

The field under this rule may have alpha-numeric characters, as well as dashes and underscores.

in:value_1,value_2,...

The field under this rule must be included in the given list of values.

not_in:value_1,value_2,...

The field under this rule must not be included in the given list of values.

min:number

The field under this rule must have a size greater or equal than the given number.

For string data, value corresponds to the number of characters. For numeric data, value corresponds to a given integer value. For an array, size corresponds to the count of the array.

max:number

The field under this rule must have a size lower or equal than the given number. Value size calculated in same way like `min` rule.

between:min,max

The field under this rule must have a size between min and max params. Value size calculated in same way like `min` and `max` rule.

url

The field under this rule must be valid url format.

ip

The field under this rule must be valid ipv4 or ipv6.

ipv4

The field under this rule must be valid ipv4.

ipv6

The field under this rule must be valid ipv6.

array

The field under this rule must be array.

same:another_field

The field value under this rule must be same with `another_field` value.

regex:/your-regex/

The field under this rule must be match with given regex.

date:format

The field under this rule must be valid date format. Parameter `format` is optional, default format is `Y-m-d`.

accepted

The field under this rule must be one of `'on'`, `'yes'`, `'1'`, `'true'`, or `true`.

present

The field under this rule must be exists, whatever the value is.

different:another_field

Opposite of `same`. The field value under this rule must be different with `another_field` value.

after:tomorrow

Anything that can be parsed by `strtotime` can be passed as a parameter to this rule. Valid examples include :

- after:next week
- after:2016-12-31
- after:2016
- after:2016-12-31 09:56:02

before:yesterday

This also works the same way as the after rule. Pass anything that can be parsed by `strtotime`

callback

You can use this rule to define your own validation rule. This rule can't be registered using string pipe. To use this rule, you should put Closure inside array of rules.

For example:

```
$validation = $validator->validate($_POST, [
    'even_number' => [
        'required',
        function ($value) {
            // false = invalid
            return (is_numeric($value) AND $value % 2 === 0);
        }
    ]
]);
```

You can set invalid message by returning a string. For example, example above would be:

```
$validation = $validator->validate($_POST, [
    'even_number' => [
        'required',
        function ($value) {
            if (!is_numeric($value)) {
                return ":attribute must be numeric.";
            }
            if ($value % 2 !== 0) {
                return ":attribute is not even number.";
            }
            // you can return true or don't return anything if value is valid
        }
    ]
]);
```

> Note: `Rakit\Validation\Rules\Callback` instance is binded into your Closure. So you can access rule properties and methods using `$this`.

Register/Modify Rule

Another way to use custom validation rule is to create a class extending `Rakit\Validation\Rule`. Then register it using `setValidator` or `addValidator`.

For example, you want to create `unique` validator that check field availability from database.

First, lets create `UniqueRule` class:

```
<?php

use Rakit\Validation\Rule;

class UniqueRule extends Rule
{
    protected $message = ":attribute :value has been used";

    protected $fillable_params = ['table', 'column', 'except'];

    protected $pdo;

    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }

    public function check($value)
    {
        // make sure required parameters exists
        $this->requireParameters(['table', 'column']);

        // getting parameters
        $column = $this->parameter('column');
        $table = $this->parameter('table');
        $except = $this->parameter('except');
        if ($except AND $except == $value) {
            return true;
        }

        // do query
        $stmt = $this->pdo->prepare("select count(*) as count from `{$table}` where `{$column}` = :value");
        $stmt->bindParam(':value', $value);
        $stmt->execute();
        $data = $stmt->fetch(PDO::FETCH_ASSOC);

        // true for valid, false for invalid
        return intval($data['count']) === 0;
    }
}
```

Then you need to register `UniqueRule` instance into validator like this:

```
use Rakit\Validation\Validator;

$validator = new Validator;

$validator->addValidator('unique', new UniqueRule($pdo));
```

Now you can use it like this:

```
$validation = $validator->validate($_POST, [
    'email' => 'email|unique:users,email,exception@mail.com'
]);
```

In `UniqueRule` above, property `$message` is used for default invalid message. And property `$fillable_params` is used for `fillParameters` method (defined in `Rakit\Validation\Rule` class). By default `fillParameters` will fill parameters listed in `$fillable_params`. For example `unique:users,email,exception@mail.com` in example above, will set:

```
$params['table'] = 'users';
$params['column'] = 'email';
$params['except'] = 'exception@mail.com';
```

> If you want your custom rule accept parameter list like `in`, `not_in`, or `uploaded_file` rules, you just need to override `fillParameters(array $params)` method in your custom rule class.

Note that `unique` rule that we created above also can be used like this:

```
$validation = $validator->validate($_POST, [
    'email' => [
        'required', 'email',
        $validator('unique', 'users', 'email')->message('Custom message')
    ]
]);
```

So you can improve `UniqueRule` class above by adding some methods that returning its own instance like this:

```
<?php

use Rakit\Validation\Rule;

class UniqueRule extends Rule
{
    ...

    public function table($table)
    {
        $this->params['table'] = $table;
        return $this;
    }

    public function column($column)
    {
        $this->params['column'] = $column;
        return $this;
    }

    public function except($value)
    {
        $this->params['except'] = $value;
        return $this;
    }

    ...
}
```

Then you can use it in more funky way like this:

```
$validation = $validator->validate($_POST, [
    'email' => [
        'required', 'email',
        $validator('unique')->table('users')->column('email')->except('exception@mail.com')->message('Custom message'
    ]
]);
```