Laravel Crud Tools

Release 0.0.61

Thiago Przyczynski

Apr 05, 2022

CONTENTS:

Easy to use Laravel CRUD package with Controller, Model and Log system built in.

CHAPTER

ONE

TABLE OF CONTENTS

CHAPTER

TWO

INSTALLATION

Install through composer using: `composer install thiagoprz\crud-tools`

If you don't have package auto discovery enabled add CrudToolsServiceProvider to your `config/app.php`:

```
...
'providers' => [
    ...
    \Thiagoprz\CrudTools\CrudToolsServiceProvider::class,
],
...
```

Publish Crud Tools service provider to allow stubs customization:

`php artisan vendor:publish --provider="Thiagoprz\CrudTools\CrudToolsServiceProvider"`

CHAPTER

THREE

USAGE

3.1 CRUD Controller:

A CRUD Controller can be achieve by just creating a standard controller class using ControllerCrud trait.

The next step is to create a folder inside `resources/views` with the desired namespace or on root folder if the controller won't be using a specific namespace (admin on the example).

```
<?php

namespace App\Http\Controllers\Admin;

use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Thiagoprz\CrudTools\Http\Controllers\ControllerCrud;

class UserController extends Controller
{
    use ControllerCrud;
    public $modelClass = User::class;
}
```

Views directory structure used by Controller CRUD based on the above example:

Folder: > views/admin/user

Files: > create.blade.php

edit.blade.php

Available vars: $model (the model being updated)

form.blade.php

Available vars: $model (the model being updated - only on edit action)

index.blade.php

Available vars: $items (the pagination object containing a filtered collection of the model)

show.blade.php

Available vars: $model (the model being displayed)

3.2 CRUD Model:

For models you just need to add the trait ModelCrud and after that create a static property declaring model's validations (based on the create, update and/or delete scenarios), default order, filtering rules, upload file rules, define resources, and with / countable relationships.

- Validations:

```
<?php
...
use Thiagoprz\CrudTools\Models\ModelCrud;
class User extends Authenticatable
{
    use ModelCrud;

/**
 * Model validations
 *
 * @var array
 */
static $validations = [
    'create' => [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ],
    'update' => [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ],
];
...
}
```

- Searchable fields:

You can create a $searchable property that will hold fields allowed to be searched on the static method **search()** - very useful with the ControllerCrud.

```
<?php
...
use Thiagoprz\CrudTools\Models\ModelCrud;
class User extends Authenticatable
{
    use ModelCrud;
    /**
     * Fields that can be searched by (static)method search()
     *
     * @var array
     */
    static $searchable = [
        'id' => 'int',
        'name' => 'string',
```

(continues on next page)

(continued from previous page)

```
        'created_at' => 'datetime',
    ];
    ...
}
```

- Range searchable fields:

Types available: int, string, date, datetime and decimal.

You can use input filters using *"from" and* "to" suffix on date, datetime and decimal fields:

```
<!-- Filtering created_at usig field "from" ( where created_at >= $created_at_from ) -->
<label>Period from: </label>
<input type="date" name="created_at_from">

<!-- Filtering created_at usig field "to" ( where created_at <= $created_at_to ) -->
<label>To:</label>
<input type="date" name="created_at_to">
```

Type	Description	Suffixes: *from *to
int	Integer fields, can be used to search a range of records by using *"from" and* "to" suffixes	Yes
decimal	Float, Double, Real or any decimal type of field. *"from" and* "to" suffixes allowed	Yes
string	Any string field to be search using "WHERE field LIKE '%SEARCH%'"	No
string	Any string field to be search using "WHERE field = 'SEARCH'"	No
date-time	Datetime and Timestamp fields	Yes
date	Date fields	Yes

- Custom searchable field methods:

In addition to use standard search based on type of fields you can add your on custom methods to customize search of specific fields. Create a method called "**searchField**" where Field is the name of the field with only first letter upper case.

Example:

```
<?php
...
use Thiagoprz\CrudTools\Models\ModelCrud;
class Books extends Model
{
    ...

    /**
     * Searching only by the start of the title of the book with LIKE
     */
    public static function searchTitle($query, $title)
    {
        $query->where('title', 'LIKE', "$title%");
    }

}
```

- Sortable fields:

You can define the fields that will be used as default sorting of your model on the index action. Also, you can pass an "order" input used by the search method allowing the override the default order defined by this variable.

```
<?php
...
use Thiagoprz\CrudTools\Models\ModelCrud;
class Books extends Model
{
    use ModelCrud;
    /**
     * Default order
     *
     * @var array
     */
    static $search_order = [
        'title' => 'ASC',
        'updated_at' => 'DESC',
        'created_at' => 'DESC',
    ];
    ...
}
```

- Upload fields:

You can create a fileUploads method to define which and where your uploadable fields will store the files:

```
<?php
...
use Thiagoprz\CrudTools\Models\ModelCrud;
class User extends Authenticatable
{
    use ModelCrud;
    ...
    /**
     * @param Campaign $model
     * @return array
     */
    public static function fileUploads(Campaign $model)
    {
        return [
            'FIELD_NAME' => [
                'path' => 'FOLDER', // Mandatory
                'name' => 'FILE_NAME', // (OPTIONAL)if not provided will be the file␣
↪original name
            ],
        ];
    }
    ...
}
```

CHAPTER

FOUR

CRUD GENERATORS

4.1 Controller Generator:

You can create a standard Controller to work with a model by using the following command:

`php artisan make:crud-controller NAMESPACE1/NAMEController NAMESPACE2/Model`

NAMESPACE1: Controller's namespace

NAMEController: is the name of the controller

NAMESPACE2: Model's namespace

Model: Name of the model

4.2 Model Generator:

To easily create a model with all Crud Tools enabled use:

```
php artisan make:crud-model NAMESPACE/Model

NAMESPACE: Model's namespace Model: Name of the model
```

- Available options
- **–fillable**: comma separated fields for fillable attributes
- **–searchable**: comma separated fields for searchable attributes (based on search() method)
- **–primaryKey**: field or comma separated fields that are the table's primary key
- **–softDeletes**: if passed enables SoftDeletes trait on class
- **–uploads**: if passed adds fileUploads() method on class
- **–logable**: adds Logable trait on model

CHAPTER

FIVE

ENABLING LOGS

To enable automatic logs on your models you need to publish Spatie Activity Logger migrations:

```
php artisan vendor:publish --provider="Spatie\Activitylog\ActivitylogServiceProvider"
--tag="migrations"
```

Run migrations:

```
php artisan migrate
```

For more information you can read Spatie Activity Log Documentations.

CHAPTER

SIX

SUPPORT

6.1 Issues

Please feel free to indicate any issues on this packages, it will help a lot. I will address it as soon as possible.

6.2 Supported By Jetbrains

This project is being developed with the help of Jetbrains through its project to support Open Source software.

6.3 Buy me a Coffee

CHAPTER

SEVEN

INDICES AND TABLES

- search
- genindex