ChristianKuri / laravel-favorite

github.com/ChristianKuri/laravel-favorite

ChristianKuri

Laravel Favorite (Laravel 5, 6 Package)

Allows Laravel Eloquent models to implement a 'favorite' or 'remember' or 'follow' feature.

Index

Installation

1. Install the package via Composer

```
$ composer require christiankuri/laravel-favorite
```

2. In Laravel >=5.5 this package will automatically get registered. For older versions, update your `config/app.php` by adding an entry for the service provider.

```
'providers' => [
    // ...
    ChristianKuri\LaravelFavorite\FavoriteServiceProvider::class,
];
```

3. Publish the database from the command line:

```
php artisan vendor:publish --provider="ChristianKuri\LaravelFavorite\FavoriteServiceProvider"
```

4. Migrate the database from the command line:

```
php artisan migrate
```

Models

Your User model should import the `Traits/Favoriteability.php` trait and use it, that trait allows the user to favorite the models. (see an example below):

```
use ChristianKuri\LaravelFavorite\Traits\Favoriteability;

class User extends Authenticatable
{
 use Favoriteability;
}
```

Your models should import the `Traits/Favoriteable.php` trait and use it, that trait have the methods that you'll use to allow the model be favoriteable. In all the examples I will use the Post model as the model that is 'Favoriteable', thats for example propuses only. (see an example below):

```
use ChristianKuri\LaravelFavorite\Traits\Favoriteable;

class Post extends Model
{
   use Favoriteable;
}
```

That's it ... your model is now **"favoriteable"**! Now the User can favorite models that have the favoriteable trait.

Usage

The models can be favorited with and without an authenticated user (see examples below):

Add to favorites and remove from favorites:

If no param is passed in the favorite method, then the model will asume the auth user.

```
$post = Post::find(1);
$post->addFavorite(); // auth user added to favorites this post
$post->removeFavorite(); // auth user removed from favorites this post
$post->toggleFavorite(); // auth user toggles the favorite status from this post
```

If a param is passed in the favorite method, then the model will asume the user with that id.

```
$post = Post::find(1);
$post->addFavorite(5); // user with that id added to favorites this post
$post->removeFavorite(5); // user with that id removed from favorites this post
$post->toggleFavorite(5); // user with that id toggles the favorite status from this post
```

The user model can also add to favorites and remove from favrites:

```
$user = User::first();
$post = Post::first();
$user->addFavorite($post); // The user added to favorites this post
$user->removeFavorite($post); // The user removed from favorites this post
$user->toggleFavorite($post); // The user toggles the favorite status from this post
```

Return the favorite objects for the user:

A user can return the objects he marked as favorite. You just need to pass the **class** in the `favorite()` method in the `User` model.

```
$user = Auth::user();
$user->favorite(Post::class); // returns a collection with the Posts the User marked as favorite
```

Return the favorites count from an object:

You can return the favorites count from an object, you just need to return the `favoritesCount` attribute from the model

```
$post = Post::find(1);
$post->favoritesCount; // returns the number of users that have marked as favorite this object.
```

Return the users who marked this object as favorite

You can return the users who marked this object, you just need to call the `favoritedBy()` method in the object

```
$post = Post::find(1);
$post->favoritedBy(); // returns a collection with the Users that marked the post as favorite.
```

Check if the user already favorited an object

You can check if the Auth user have already favorited an object, you just need to call the `isFavorited()` method in the object

```
$post = Post::find(1);
$post->isFavorited(); // returns a boolean with true or false.
```

Testing

The package have integrated testing, so everytime you make a pull request your code will be tested.

Change log

Please see CHANGELOG for more information on what has changed recently.

Contributions

Contributions are **welcome** and will be fully **credited**. We accept contributions via Pull Requests on Github.

Pull Requests

- **PSR-2 Coding Standard** - Check the code style with `$ composer check-style` and fix it with `$ composer fix-style` .
- **Add tests!** - Your patch won't be accepted if it doesn't have tests.

- **Document any change in behaviour** - Make sure the `README.md` and any other relevant documentation are kept up-to-date.
- **Consider our release cycle** - We try to follow SemVer v2.0.0. Randomly breaking public APIs is not an option.
- **Create feature branches** - Don't ask us to pull from your master branch.
- **One pull request per feature** - If you want to do more than one thing, send multiple pull requests.
- **Send coherent history** - Make sure each individual commit in your pull request is meaningful. If you had to make multiple intermediate commits while developing, please squash them before submitting.

Security

Please report any issue you find in the issues page. Pull requests are welcome.

Credits

- Christian Kuri
- All Contributors

License

The MIT License (MIT). Please see License File for more information.