tuts+

CODE > LARAVEL 5

The Repository Pattern in Laravel 5

by Alireza Rahmani Khalili 8 Mar 2016

Difficulty: Advanced Length: Short Languages: English

Laravel 5 PHP Web Development Design Patterns Web Apps

The repository pattern was introduced for the first time by Eric Evans in his Domain-Driven Design book. The repository is, in fact, the entry point for the *application* to access the *domain* layer.

To put it simply, the repository allows all your code to use objects without having to know how the objects are persisted. The repository contains all the knowledge of persistence, including mapping from tables to objects. This provides a more object-oriented view of the persistence layer and makes the mapping code more encapsulated.

The only way to make your repositories work in Laravel (as a real repository—Eric Evans Domain-Driven Design book) is to change the default ORM from active record to data mapper. The best substitute is Doctrine.

The Doctrine ORM

Doctrine is an ORM (object-relational mapping) which implements the data mapper pattern and allows you to make a clean separation of the application's business rules from the persistence layer of the database. Doctrine uses DQL, rather than SQL. DQL brings you object query language, meaning that instead of a traditional relational query term, you would have queries in object term.

It allows you to write the database queries in an object-oriented way and helps when you need to query the database in a way which can't be achieved using the default repository methods. In my opinion, DQL is the most powerful way to keep in touch with your database.

Doctrine vs. Eloquent

Doctrine entities are just a plain PHP simple class and do not add overhead to any ORM inheritance. Doctrine manages your multiple query requests with the same inheritance without hitting the database, meaning the entity object exists for the entire request.

The other nice feature of Doctrine is that instead of migrating files to create the database schema, the database is automatically created to reflect the meta data in the entity annotations. On the other hand, Eloquent is less complicated and very easy to use.

A complete comparison between these two would necessitate a separate article. As you can see, a Doctrine object is lighter and more abstract. However, Doctrine will only fit specific projects, so it may bring you overhead at times. I believe it depends on the programmer to choose the best ORM for the app.

The Blog App

Now it's time to create a blog app with Laravel. First, we need to set up Doctrine. There is a bridge to allow for matching with Laravel 5's existing configuration. To install Doctrine 2 within our Laravel project, we run the following command:

`bash composer require laravel-doctrine/orm`

As usual, the package should be added to the `app/config.php`, as the service provider:

`php LaravelDoctrine\ORM\DoctrineServiceProvider::class,`

The alias should also be configured:

`php 'EntityManager' => LaravelDoctrine\ORM\Facades\EntityManager::class`

Finally, we publish the package configuration with:

`bash php artisan vendor:publish --tag="config"`

Now we're done here.

Entities are important parts of the app `App\Entities\Post.php`:

```php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
```

/** * @ORM\Entity * @ORM\Table(name="posts") * @ORM\HasLifecycleCallbacks() */ class Post { /** * @var integer $id * @ORM\Column(name="id", type="integer", unique=true, nullable=false) * @ORM\Id * @ORM\GeneratedValue(strategy="AUTO") * */ private $id;

```
/**
 * @ORM\Column(type="string")
 */
private $title;

/**
 * @ORM\Column(type="text")
 */
private $body;

public function __construct($input)
{
    $this->setTitle($input['title']);
    $this->setBody($input['body']);
}

public function setId($id)
{
    return $this->id=$id;
}

public function getId()
{
    return $this->id;
}

public function getTitle()
{
    return $this->title;
}

public function setTitle($title)
{
    $this->title = $title;
}

public function getBody()

{
    return $this->body;
}
```

````
public function setBody($body)
{
    $this->body = $body;
} } ```
````

Now it's time to create the *Repository*, which was described earlier. `App/Repositories/PostRepo.php` :

```php namespace App\Repository;

use App\Entity\Post; use Doctrine\ORM\EntityManager; class PostRepo {

````
/**
 * @var string
 */
private $class = 'App\Entity\Post';
/**
 * @var EntityManager
 */
private $em;

public function __construct(EntityManager $em)
{
    $this->em = $em;
}

public function create(Post $post)
{
    $this->em->persist($post);
    $this->em->flush();
}

public function update(Post $post, $data)
{
    $post->setTitle($data['title']);
    $post->setBody($data['body']);
    $this->em->persist($post);
    $this->em->flush();
}

public function PostOfId($id)
{
    return $this->em->getRepository($this->class)->findOneBy([
        'id' => $id
````
````

```
    ]);
}

public function delete(Post $post)
{
    $this->em->remove($post);
    $this->em->flush();
}

/**
 * create Post
 * @return Post
 */
private function perpareData($data)
{
    return new Post($data);
} }
```

~~~

The controller: `App/Http/Controllers/PostController.php`:

```php namespace App\Http\Controllers; use App\Repository\PostRepo as repo; use App\Validation\PostValidator;

class PostController extends Controller { private $repo;

~~~
public function __construct(repo $repo)
{
    $this->repo = $repo;
}

public function edit($id=NULL)
{
    return View('admin.edit')->with(['data' => $this->repo->postOfId($id)]);
}

public function editPost()
{
    $all = Input::all();
    $validate = PostValidator::validate($all);
    if (!$validate->passes()) {

        return redirect()->back()->withInput()->withErrors($validate);
    }
    $Id = $this->repo->postOfId($all['id']);
    if (!is_null($Id)) {
```
```

```
            $this->repo->update($Id, $all);
            Session::flash('msg', 'edit success');
        } else {
            $this->repo->create($this->repo->perpare_data($all));
            Session::flash('msg', 'add success');
        }
        return redirect()->back();
    }

    public function retrieve()
    {
        return View('admin.index')->with(['Data' => $this->repo->retrieve()]);
    }

    public function delete()
    {
        $id = Input::get('id');
        $data = $this->repo->postOfId($id);
        if (!is_null($data)) {
            $this->repo->delete($data);
            Session::flash('msg', 'operation Success');
            return redirect()->back();
        } else {
            return redirect()->back()->withErrors('operationFails');
        }
    } } ```
```

As you see, I've used the Flash helper to manage the messages (you can use Laravel's). Regarding the Validator, I should add that you can create your own (as I do) or use the Laravel default, depending on your preference.

View files are the same as usual. In this sample view, the file seems like `resources/views/admin/edit.blade.php`:

```php

@if (Session::has('flash_notification.message'))
× {!! Session::get('flash_notification.message') !!}
@endif
@if($errors->has())
@foreach ($errors->all() as $error)

- {!! $error !!}

@endforeach
@endif
{!! 'title' !!}

`{!! is_object($ListData)?$List`
```

{!! 'Body' !!}

```
{!!
is_object($ListData
```

{!! 'save' !!}

```
    </div>
</div> ```
```

The routing and other operations would be as usual.

Conclusion

Now you see how you can easily create a repository based on Doctrine in Laravel 5.0, which will result in many benefits.

For those of you who are either just getting started with Laravel or looking to expand your knowledge, site, or application with extensions, we have a variety of things you can study in Envato Market.

Alireza Rahmani Khalili

To whom it may concern, I'm Alireza a Software Architect, Consultant and Trainer of Enterprise Web Applications. For a significant chunk of my waking hours I'm a PHP expert, Author, Speaker.

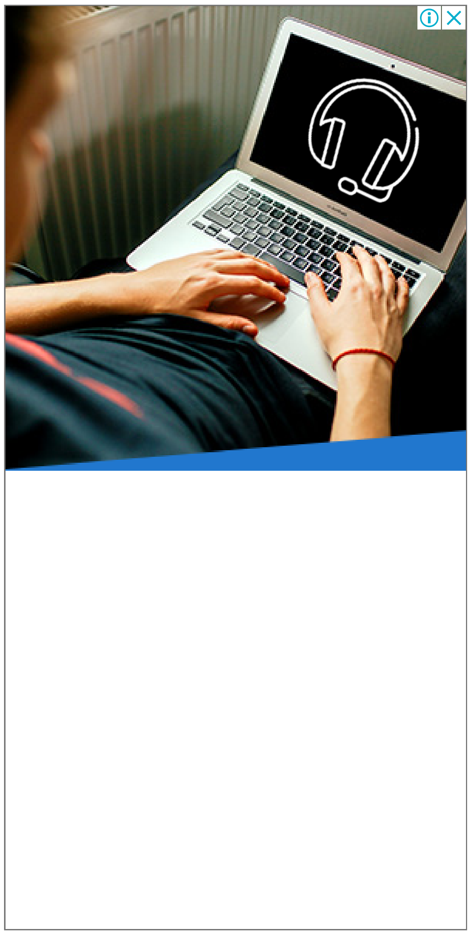
FEED LIKE FOLLOW FOLLOW

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Email Address

Update me weekly



Advertisement

View on GitHub

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native

11 Comments Tuts+ Hub 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Mike Hopley • 3 years ago

Anyone thinking about using the repository pattern -- at least in this "purist" way -- should first have a listen to this episode of the Laravel podcast: http://www.laravelpodcast.c...

Repositories are useful for storing repeated queries. But the purist approach of using them to decouple your application from ActiveRecord / Eloquent is fraught with pitfalls. Essentially you cannot have the convenience of ActiveRecord without coupling yourself to it.

Datamapper itself doesn't inherently solve all these issues. You still have a similar tension between "I want to decouple my code from the ORM" and "I want to use the ORM". It's your choice, but for the vast majority of projects you are better off just using the ORM and not obsessing over "perfect architecture".

2 ^ | v • Reply • Share ›

Braj Mohan ➦ Mike Hopley • 3 years ago

I always wondered if I am the only one who think this as over architecture. I tried several things in past by reading these tutorials but always failed miserably by end up with extra nonsense layer of complexity which is difficult to maintain and justify to other developers in team.

But after reading "not obsessing over "perfect architecture"" I got my confidence back. Thank you

1 ^ | v • Reply • Share ›

Mike Hopley ➦ Braj Mohan • 3 years ago

No problem. :) This was one of the things that annoyed me back when I was learning Laravel at the height of the repository fad. I toyed around with repositories a little before realising I was getting negative value from them when used in this way.

^ | v • Reply • Share ›

Vladislav Rastrusny ➦ Mike Hopley • 2 years ago

> but for the vast majority of projects you are better off just using the ORM and not obsessing over "perfect architecture".

I guess this primarily depends on the size of the project. On small to medium projects you can inject AR model into a controller without any problems. But on large projects, which are developed by several teams such an action when used inaccurately may lead to code mess.

^ | v • Reply • Share ›

Someone • 2 years ago

This article does not explain the repository pattern, it only shows how you can create custom repositories by using Doctrine build in repository.
A repository must be seen as a collection of objects and before query the database it looks at the repository/collection for that item.
Let's say we have a user repository:

```
class UserRepository
{
public function __construct() {
$this->data = []; //our "collection"
}
public function save($name, $email) {
$user = new User($name, $email);
$this->data[$name] = $user;
SomeDb::save($user); // actual db query
}
public function getUserByName($name) {
if ( array_key_exists($name, $this->data[$name]) ) {
return $this->data[$name];
```

see more

Reply • Share ›

Nikola Poša • 2 years ago

There is also nice article: https://medium.com/laravel-... on how to implement DataMapper-like architecture with Eloquent.

Reply • Share ›

Andrian Motelica • 3 years ago

tip for entity. to reduce complexity of getters/setters a Trait could be used in each model setting 2 "magic methods"

```
public function __get($property) {
$customGetter = "get".ucfirst($property);
return $this->$customGetter();
}
```

same for setter

```
public function __set($property, $value){
$customSetter = "set".ucfirst($property);
return $this->$customSetter($value);
}
```

only setter the entities will have set is

```
public function setId($id) {
throw new Exception("You can't set this property.);
}
```

see more

Reply • Share ›

Povilas Korop • 3 years ago

This article explains the HOW part, but more and more I wonder about the WHY about repositories. Especially in Laravel, where Eloquent is already kind of a repository.

Reply • Share ›

Mike Hopley ➦ Povilas Korop • 3 years ago

The idea is that you could, for example, switch out your data access layer for something completely different instead of Eloquent. In some projects, this is actually useful (or even essential).

But for the vast majority of projects, it is counter-productive. Very few projects benefit from this degree of flexibility. Flexibility in code is not inherently good, because it comes with a cost as well as a benefit, and often the cost is steep (and easily underestimated).

There's nothing wrong with the article. In fact I think the author strikes a measured and sensible tone. The problem is the hordes of smug coders shouting that, if you're using Eloquent in your controllers, you're Doing It Wrong.

(Also note that repositories have other uses, namely storing more complex queries that you might not want cluttering up your controllers again and again.)

1 ^ | v • Reply • Share ›

ably ➦ Mike Hopley • 2 years ago

I do not agree that there's nothing wrong with this article. It's chaotic and the code is very messy...
Simple example. This part:

```
public function editPost()
{
$all = Input::all();
$validate = PostValidator::validate($all);
if (!$validate->passes()) {
return redirect()->back()->withInput()->withErrors($validate);
}
$Id = $this->repo->postOfId($all['id']);
if (!is_null($Id)) {
$this->repo->update($Id, $all);
Session::flash('msg', 'edit success');
} else {
$this->repo->create($this->repo->perpare_data($all));
Session::flash('msg', 'add success');
}
```

see more

^ | v • Reply • Share ›

kimball ➦ Povilas Korop • 3 years ago

Doctrine supports a wider range of databases than eloquent (db2 for example). The repository pattern makes sense in this matter. Jeffrey Way talks about repositories and programming to an interface in some of his videos. It is definitely more coding required.

^ | v • Reply • Share ›

QUICK LINKS - Explore popular categories

ENVATO TUTS+ ✚

JOIN OUR COMMUNITY ✚

HELP ✚

tuts+

26,484	1,167	28,586
Tutorials	Courses	Translations