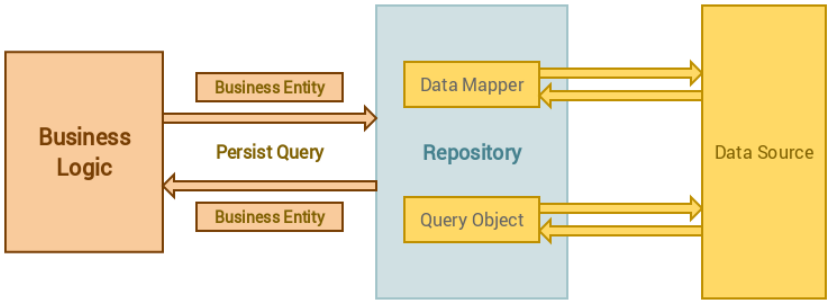
Connor Leech Follow
I work at Stitch Labs and build https://employbl.com/
Feb 10 · 4 min read

Use the Repository Design pattern in a Laravel application



I previously wrote about how to build a task app with Laravel and Vue.js. Laravel is a PHP framework for building scalable web applications and APIs. Vue.js is a Javascript framework and alternative to jQuery or React.

Build a Task List with Laravel 5.4 and Vue 2

We're going to follow along with @felicianopj post that is available here. The source code for this...

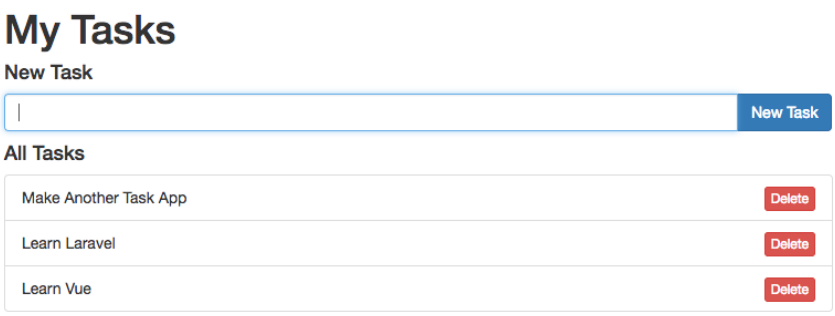
medium.com

Get Started

In this tutorial we're going to add functionality to the application we built in the previous tutorial. Clone the github repo, run composer install, npm install and connect to you're database.

```
$ mysql -uroot -p
mysql> create database laravelTaskApp;
```

If you're stuck at this point check out other articles I've written about installing MySQL and initial Laravel setup. At **localhost:8000** you can see an app that asynchronously adds and deletes tasks. That means it does the operations and displays the up to date data without refreshing the webpage.



The task app we built in the previous tutorial.

The Repository Design Pattern

In the previous tutorial we wrote all of our application logic in the controller. There's an alternative approach to development that abstracts some calls into PHP classes called Repositories. The idea is that we can decouple models from controllers and assign readable names to complicated queries.

We're going to refactor our app to use the Repository Pattern. The first step is to create a file for **app/Repositories/Repository.php.**

```
<?php namespace App\Repositories;

use Illuminate\Database\Eloquent\Model;

class Repository implements RepositoryInterface
{
    // model property on class instances
    protected $model;

    // Constructor to bind model to repo
```

```
    public function __construct(Model $model)
    {
        $this->model = $model;
    }

    // Get all instances of model
    public function all()
    {
        return $this->model->all();
    }

    // create a new record in the database
    public function create(array $data)
    {
        return $this->model->create($data);
    }

    // update record in the database
    public function update(array $data, $id)
    {
        $record = $this->find($id);
        return $record->update($data);
    }

    // remove record from the database
    public function delete($id)
    {
        return $this->model->destroy($id);
    }

    // show the record with the given id
    public function show($id)
    {
        return $this->model-findOrFail($id);
    }

    // Get the associated model
    public function getModel()
    {
        return $this->model;
    }

    // Set the associated model
    public function setModel($model)
    {
        $this->model = $model;
        return $this;
    }

    // Eager load database relationships
    public function with($relations)
    {
        return $this->model->with($relations);
    }
}
```

This file defines our Repository class. Instances of this class have a model property that we tie to an Eloquent model. Once this is bound in the constructor we can call Eloquent methods like `findOrFail`, `update` or `all` from the class methods.

The **implements RepositoryInterface** section isn't strictly necessary but it adds an extra layer of structure to our code. An interface is a contract that defines the methods a class MUST have defined. In our case the interface looks like this:

```
<?php namespace App\Repositories;

interface RepositoryInterface
{
    public function all();

    public function create(array $data);

    public function update(array $data, $id);

    public function delete($id);

    public function show($id);
}
```

If we make new Repositories that implement this interface we'll always know these methods are defined. Interfaces provide structure so we know what our code needs to do.

Back in our **TaskController.php** file we instantiate a repository and pass in the Task model to it.

```
<?php

namespace App\Http\Controllers;

use App\Task;
use App\Repositories\Repository;
use Illuminate\Http\Request;

class TaskController extends Controller
{
    // space that we can use the repository from
   protected $model;
```

```
    public function __construct(Task $task)
    {
        // set the model
        $this->model = new Repository($task);
    }

    public function index()
    {
        return $this->model->all();
    }

    public function store(Request $request)
    {
        $this->validate($request, [
            'body' => 'required|max:500'
        ]);

        // create record and pass in only fields that are
fillable
        return $this->model->create($request->only($this-
>model->getModel()->fillable));
    }

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

    public function update(Request $request, $id)
    {
        // update model and only pass in the fillable
fields
        $this->model->update($request->only($this->model-
>getModel()->fillable), $id);

        return $this->model->find($id);
    }

    public function destroy($id)
    {
        return $this->model->delete($id);
    }
}
```

The Laravel service container will automatically resolve our dependencies and inject them into the controller instance (docs).

At this point our application works exactly the same but our code has been refactored to use repositories and we've added a couple more API endpoints.

Source code available on Github

. . .

 Create a free profile on Employbl.com to browse tech companies in San Francisco