LARAVEL

25 Laravel Tips and Tricks

by Jeffrey Way *31 Oct 2013* *11 Comments*

57 16

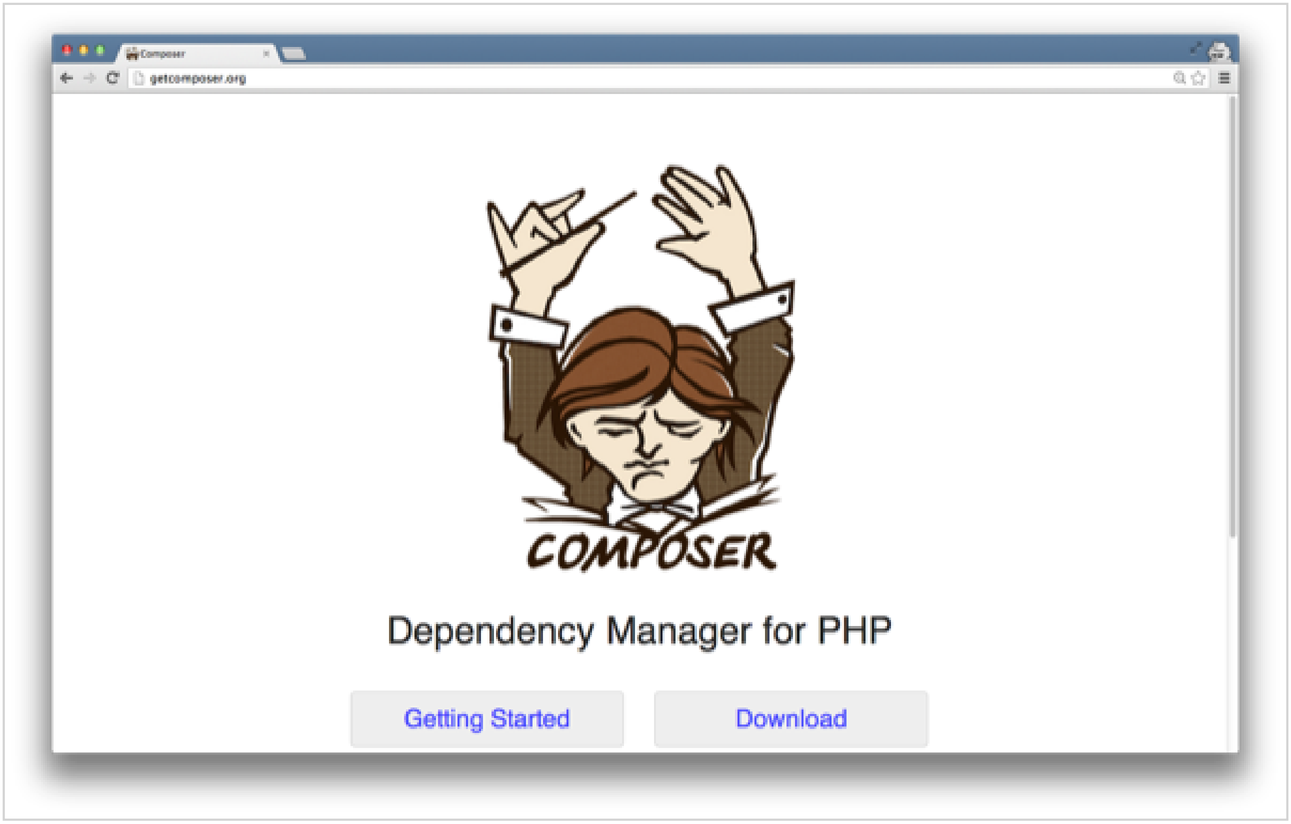
There was a period of time, not too long ago, when PHP and its community were, for lack of better words, hated. Seemingly, the headline joke of every day was one that related to how terrible PHP was. Let's see, what new PHP-slamming blog article will be posted today?

> *Yes, sadly enough, the community and ecosystem simply weren't on the same level as other modern languages.*

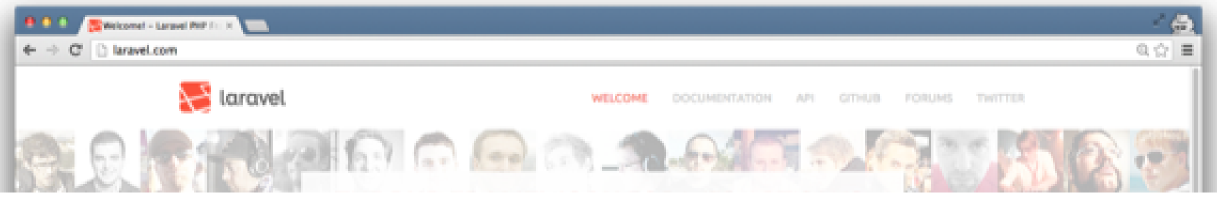
Yes, sadly enough, the community and ecosystem simply weren't on the same level as other modern languages. It seemed that PHP was destined to live out its dominating lifespan in the form of messy WordPress themes.

But, then, quite amazingly, things began to change - and quickly, too. Like a witch stirring the pot, innovative new projects began popping out of nowhere. Perhaps most notable of these projects was Composer: PHP's definitive dependency manager (not unlike Ruby's Bundler or Node's NPM). While, in the past, PHP developers were forced to wrangle PEAR into shape (a nightmare, indeed), now, thanks to Composer, they can simply update a JSON file, and immediately pull in their desired dependency. A profiler here, a testing framework there... all in seconds!

In the crowded PHP framework world, just as CodeIgniter began to fizzle out, Taylor Otwell's Laravel framework arose out of the ashes to become the darling of the community. With such a simple and elegant syntax, building applications with Laravel and PHP was - gasp - downright fun! Further, with version 4 of the framework leveraging Composer heavily, things finally seemed to be falling into place for the community.



Want migrations (version control for your database)? Done. How about a powerful Active-Record implementation? Sure, Eloquent will do the trick quite nicely. What about testing facilities? Of course. And routing? Most certainly. What about a highly tested HTTP layer? Thanks to Composer, Laravel can leverage many of the excellent Symfony components. When it comes right down to it, chances are, if you need it, Laravel offers it.



While PHP used to be not dissimilar from a game of Jenga - just one block and away from falling to pieces - suddenly, thanks to Laravel and Composer, the future couldn't look any brighter. So pull out some shades, and dig into all that the framework has to offer.

1. Eloquent Queries

Laravel offers one of the most powerful Active-Record implementations in the PHP world. Let's say that you have an `orders` table, along with an `Order` Eloquent model:

```php
class Order extends Eloquent {}
```

We can easily perform any number of database queries, using simple, elegant PHP. No need to throw messy SQL around the room. Let's grab all orders.

```php
$orders = Order::all();
```

Done. Or maybe, those orders should be returned in order, according to the release date. That's easy:

```
$orders = Order::orderBy('release_date', 'desc')->get();
```

What if, rather than fetching a record, we instead need to save a new order to the database. Sure, we can do that.

```
$order = new Order;
$order->title = 'Xbox One';
$order->save();
```

Finished! With Laravel, tasks that used to be cumbersome to perform are laughably simple.

2. Flexible Routing

Laravel is unique in that it can be used in a number of ways. Prefer a simpler, more Sinatra-like routing system? Sure, Laravel can offer that quite easily, using closures.

```
Route::get('orders', function()
{
    return View::make('orders.index')
        ->with('orders', Order::all());
});
```

This can prove helpful for small projects and APIs, but, chances are high that you'll require controllers for most of your projects. That's okay; Laravel can do that, too!

```
Route::get('orders', 'OrdersController@index');
```

Done. Notice how Laravel grows with your needs? This level of accomodation is what makes the framework as popular as it is today.

3. Easy Relationships

What do we do in the instances when we must define relationships? For example, a task will surely belong to a user. How might we represent that in Laravel? Well, assuming that the necessary database tables are setup, we only need to tweak the related Eloquent models.

```
class Task extends Eloquent {
    public function user()
    {
        return $this->belongsTo('User');
    }
}

class User extends Eloquent {
    public function tasks()
    {
        return $this->hasMany('Task');
    }
}
```

And, with that, we're done! Let's grab all tasks for the user with an id of 1. We can do that in two lines of code.

```
$user = User::find(1);
$tasks = $user->tasks;
```

However, because we've defined the relationship from both ends, if we instead want to fetch the user associated with a task, we can do that, too.

```
$task = Task::find(1);
$user = $task->user;
```

4. Form Model Binding

Often, it can be helpful to link a form to a model. The obvious example of this is when you wish to edit some record in your database. With form model binding, we can instantly populate the form fields with the values from the associated table row.

```
{{ Form::model($order) }}
    <div>
        {{ Form::label('title', 'Title:') }}
        {{ Form::text('title') }}
    </div>

    <div>
        {{ Form::label('description', 'Description:') }}
        {{ Form::textarea('description') }}
    </div>
{{ Form::close() }}
```

Because the form is now linked to a specific `Order` instance, the inputs will display the correct values from the table. Simple!

5. Cache Database Queries

Too many database queries, and, very quickly, your application can become like molasses. Luckily, Laravel offers a simple mechanism for caching these queries, using nothing more than a single method call.

Let's grab all `questions` from the database, but cache the query, since it's not likely that this table will be updated frequently.

```
$questions = Question::remember(60)->get();
```

That's it! Now, for the next hour of pages requests, that query will remain cached, and the database will not be touched.

6. View Composers

You'll encounter situations when multiple views require a certain variable or piece of data. A good example of this is a navigation bar that displays a list of tags.

Too keep controllers as minimal as possible, Laravel offers view composers to manage things like this.

```
View::composer('layouts.nav', function($view)
{
    $view->with('tags', ['tag1', 'tag2']);
});
```

With this piece of code, any time that the `layouts/nav.blade.php` view is loaded, it will have access to a variable, `$tags`, equal to the provided array.

7. Simple Authentication

Laravel takes a dead-simple approach to authentication. Simply pass an array of credentials, likely fetched from a login form, to `Auth::attempt()`. If the provided values match what is stored in the `users` table, the user will instantly be logged in.

```
$user = [
    'email' => 'email',
    'password' => 'password'
];

if (Auth::attempt($user))
{
    // user is now logged in!
    // Access user object with Auth::user()
}
```

What if we need to log the user out - perhaps, when a `/logout` URI is hit? That's easy, too.

```
Route::get('logout', function()
{
    Auth::logout();

    return Redirect::home();
});
```

8. Resources

Working RESTfully in Laravel has never been easier. To register a resourceful controller, simple call `Route::resource()`, like so:

```
Route::resource('orders', 'OrdersController');
```

With this code, Laravel will register eight routes.

- GET /orders
- GET /orders/:order
- GET /orders/create
- GET /orders/:order/edit
- POST /orders
- PUT /orders/:order
- PATCH /orders/:order
- DELETE /orders/:order

Further, the companion controller may be generated from the command line:

```
php artisan controller:make OrdersController
```

Within this generated controller, each method will correspond to one of the routes above. For examples, `/orders` will map to the `index` method, `/orders/create` will map to `create`, etc.

We now have the necessary power to build RESTful applications and APIs with ease.

9. Blade Templating

While, yes, PHP is by nature a templating language, it hasn't evolved to become an overly good one. That's okay, though; Laravel offers its Blade engine to fill the gap. Simply name your views with a `.blade.php` extension, and they will automatically be parsed, accordingly. Now, we can do such things as:

```
@if ($orders->count())
    <ul>
        @foreach($orders as $order)
            <li>{{ $order->title }}</li>
        @endforeach
    </ul>
@endif
```

10. Testing Facilities

Because Laravel makes use of Composer, we instantly have PHPUnit support in the framework out of the box. Install the framework and run `phpunit` from the command line to test it out.

Even better, though, Laravel offers a number of test helpers for the most common types of functional tests.

Let's verify that the home page returns a status code of 200.

```
public function test_home_page()
{
    $this->call('GET', '/');
    $this->assertResponseOk();
}
```

Or maybe we want to confirm that, when a contact form is posted, the user is redirected back to the home page with a flash message.

```
public function test_contact_page_redirects_user_to_home_page()
{
    $postData = [
        'name' => 'Joe Example',
        'email' => 'email-address',
        'message' => 'I love your website'
    ];

    $this->call('POST', '/contact', $postData);

    $this->assertRedirectedToRoute('home', null, ['flash_message'
}
```

11. Remote Component

As part of Laravel 4.1 - scheduled to be released in November, 2013 - you can easily write an Artisan command to SSH into your server, and perform any number of actions. It's as simple as using the `SSH` facade:

```
SSH::into('production')->run([
    'cd /var/www',
    'git pull origin master'
]);
```

Pass an array of commands to the `run()` method, and Laravel will handle the rest! Now, because it makes sense to execute code like this as an Artisan command, you only need to run `php artisan command:make DeployCommand`, and insert the applicable code into the command's `fire` method to rapidly create a dedicated deployment command!

12. Events

Laravel offers an elegant implementation of the observer pattern that you may use throughout your applications. Listen to native events, like `illuminate.query`, or even fire and catch your own.

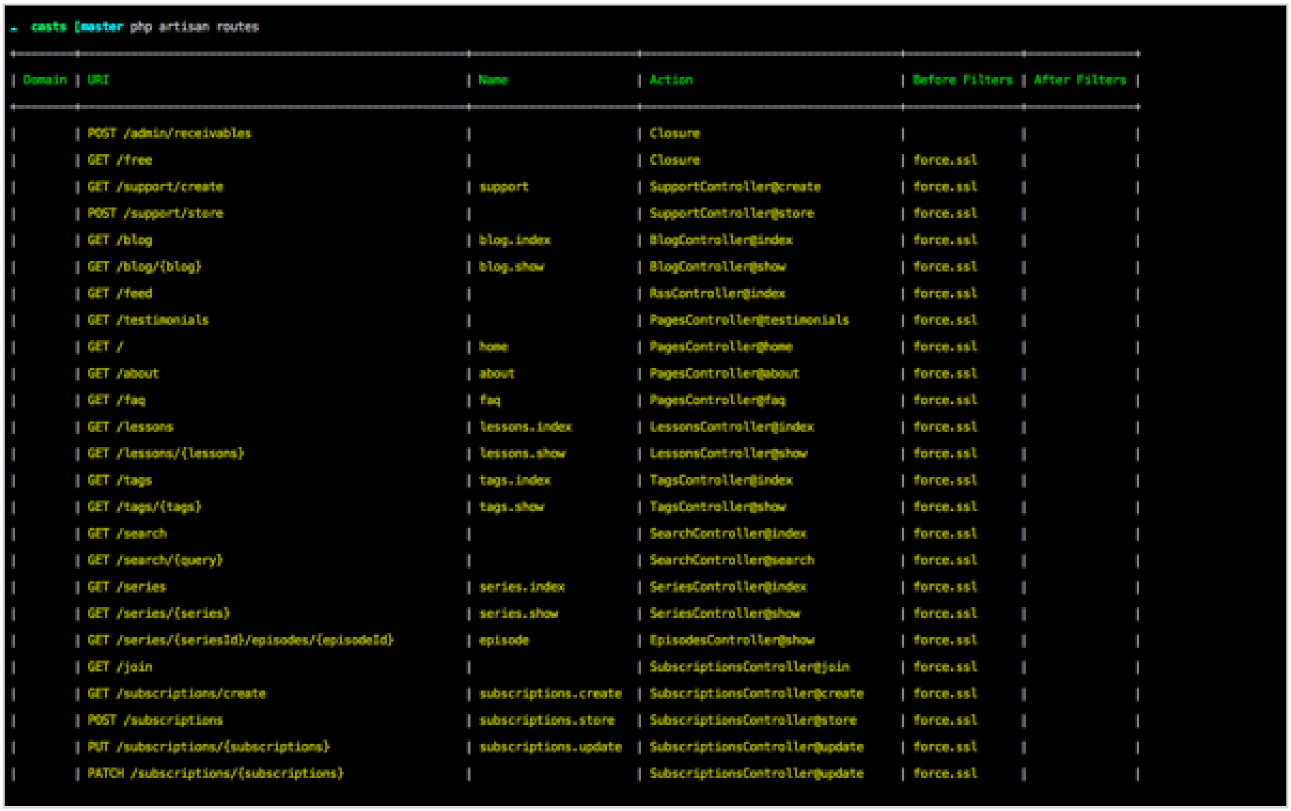
A mature use of events in an application can have an incredible effect on its maintainability and structure.

```
Event::listen('user.signUp', function()
{
    // do whatever needs to happen
    // when a new user signs up
});
```

Like most things in Laravel, if you'd instead prefer to reference a class name, rather than a closure, you can freely do so. Laravel will then resolve it out of the IoC container.

```
Event::listen('user.signUp', 'UserEventHandler');
```

13. View All Routes



```
→ casts [master php artisan routes
+--------+----------------------------------------------+-----------------------+--------------------------------+----------------+---------------+
| Domain | URI                                          | Name                  | Action                         | Before Filters | After Filters |
+--------+----------------------------------------------+-----------------------+--------------------------------+----------------+---------------+
|        | POST /admin/receivables                      |                       | Closure                        |                |               |
|        | GET /free                                    |                       | Closure                        | force.ssl      |               |
|        | GET /support/create                          | support               | SupportController@create       | force.ssl      |               |
|        | POST /support/store                          |                       | SupportController@store        | force.ssl      |               |
|        | GET /blog                                    | blog.index            | BlogController@index           | force.ssl      |               |
|        | GET /blog/{blog}                             | blog.show             | BlogController@show            | force.ssl      |               |
|        | GET /feed                                    |                       | RssController@index            | force.ssl      |               |
|        | GET /testimonials                            |                       | PagesController@testimonials   | force.ssl      |               |
|        | GET /                                        | home                  | PagesController@home           | force.ssl      |               |
|        | GET /about                                   | about                 | PagesController@about          | force.ssl      |               |
|        | GET /faq                                     | faq                   | PagesController@faq            | force.ssl      |               |
|        | GET /lessons                                 | lessons.index         | LessonsController@index        | force.ssl      |               |
|        | GET /lessons/{lessons}                       | lessons.show          | LessonsController@show         | force.ssl      |               |
|        | GET /tags                                    | tags.index            | TagsController@index           | force.ssl      |               |
|        | GET /tags/{tags}                             | tags.show             | TagsController@show            | force.ssl      |               |
|        | GET /search                                  |                       | SearchController@index         | force.ssl      |               |
|        | GET /search/{query}                          |                       | SearchController@search        | force.ssl      |               |
|        | GET /series                                  | series.index          | SeriesController@index         | force.ssl      |               |
|        | GET /series/{series}                         | series.show           | SeriesController@show          | force.ssl      |               |
|        | GET /series/{seriesId}/episodes/{episodeId}  | episode               | EpisodesController@show        | force.ssl      |               |
|        | GET /join                                    |                       | SubscriptionsController@join   | force.ssl      |               |
|        | GET /subscriptions/create                    | subscriptions.create  | SubscriptionsController@create | force.ssl      |               |
|        | POST /subscriptions                          | subscriptions.store   | SubscriptionsController@store  | force.ssl      |               |
|        | PUT /subscriptions/{subscriptions}           | subscriptions.update  | SubscriptionsController@update | force.ssl      |               |
|        | PATCH /subscriptions/{subscriptions}         |                       | SubscriptionsController@update | force.ssl      |               |
```

As an application grows, it can be difficult to view which routes have been registered. This is especially true if proper care has not been given to your `routes.php` file (i.e. abusive implicit routing).

Laravel offers a helpful `routes` command, which will display all registered routes, as well as the controller methods that they trigger.

```
php artisan routes
```

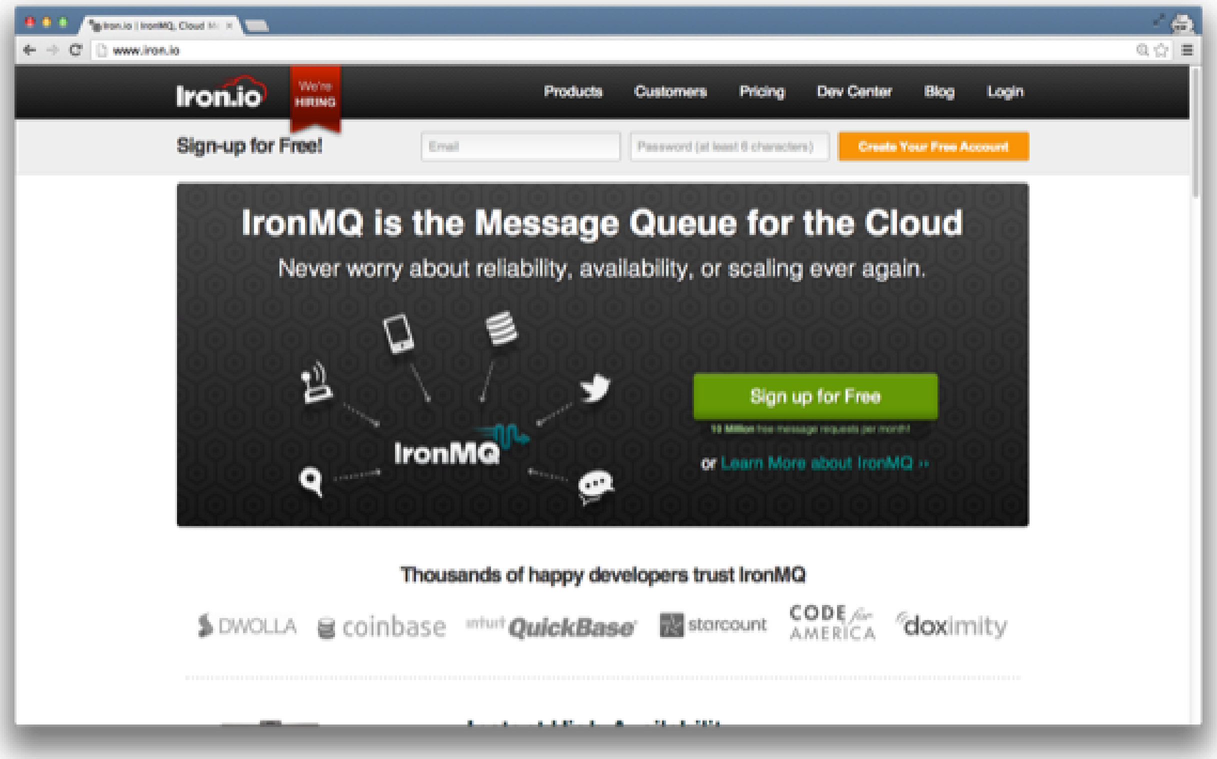
14. Queues

Think about when a user signs up for your application. Likely, a number of events must take place. A database table should be updated, a newsletter list should be appended to, an invoice must be raised, a welcome email might be sent, etc. Unfortunately, these sorts of actions have a tendency to take a long time.

Why force the user to wait for these actions, when we can instead throw them into the background?

```
Queue::push('SignUpService', compact('user'));
```

Perhaps the most exciting part, though, is that Laravel brilliantly offers support for Iron.io push queues. This means that, even without an ounce of "worker" or "daemon" experience, we can still leverage the power of queues. Simply register a URL end-point using Laravel's helpful `php artisan queue:subscribe` command, and Iron.io will ping your chosen URL each time a job is added to the queue.



Simple steps to faster performance!

15. Easy Validation

When validation is required (and when isn't it), Laravel again comes to the rescue! Using the `Validator` class is as intuitive as can be. Simply pass the object under validation, as well as a list of rules to the `make` method, and Laravel will take care of the rest.

```php
$order = [
    'title' => 'Wii U',
    'description' => 'Game console from Nintendo'
];

$rules = [
    'title' => 'required',
    'description' => 'required'
];

$validator = Validator::make($order, $rules);

if ($validator->fails())
{
    var_dump($validator->messages()); // validation errors array
}
```

Typically, this type of code will be stored within your model, meaning that validation of, say, an order could be reduced to a single method call:

```php
$order->isValid();
```

16. Tinker Tinker

Especially when first learning Laravel, it an be helpful to tinker around with the core. Laravel's `tinker` Artisan command can help with this.

> *As part of version 4.1, the `tinker` command is even more powerful, now that it leverages the popular Boris component.*

```
$ php artisan tinker

> $order = Order::find(1);
> var_dump($order->toArray());
> array(...)
```

17. Migrations

Think of migrations as version control for your database. At any given point, you may "rollback" all migrations, rerun them, and much more. Perhaps the true power rests in the power to push an app to production, and run a single `php artisan migrate` command to instantly construct your database.

To prepare the schema for a new users table, we may run:

```
php artisan migrate:make create_users_table
```

This will generate a migration file, which you may then populate according to your needs. Once complete, `php artisan migrate` will create the table. That's it! Need to roll back that creation? Easy! `php artisan migrate:rollback`.

Here's an example of the schema for a frequently asked questions table.

```
public function up()
{
    Schema::create('faqs', function(Blueprint $table) {
        $table->integer('id', true);
        $table->text('question');
        $table->text('answer');
        $table->timestamps();
    });
}

public function down()
{
    Schema::drop('faqs');
}
```

Notice how the `drop()` method performs the inverse of `up()`. This is what allows us to rollback the migration. Isn't that a lot easier that wrangling a bunch of SQL into place?

18. Generators

While Laravel offers a number of helpful generators, an incredibly helpful third-party package, called "Laravel 4 Generators," takes this even further. Generate resources, seed files, pivot tables, and migrations with schema!

In this previous tip, we were forced to manually write the schema. However, with the generators package enabled, we can instead do:

```
php artisan generate:migration create_users_table --fields="userna
```

The generator will take care of the rest. This means that, with two commands, you can prepare and build a new database table.

> *Laravel 4 Generators may be installed through Composer.*

19. Commands

As noted earlier, there are number of instances when it can be helpful to write custom commands. They can be used to scaffold applications, generate files, deploy applications, and everything in between.

Because this is such a common task, Laravel makes the process of creating a command as easy as it can be.

```
php artisan command:make MyCustomCommand
```

This command will generate the necessary boilerplate for your new custom command. Next, from the newly created, `app/commands/MyCustomCommand.php`, fill the appropriate name and description:

```
protected $name = 'command:name';
protected $description = 'Command description.';
```

And, finally, within the command class' `fire()` method, perform any action that you need to. Once finished, the only remaining step is to register the command with Artisan, from `app/start/Artisan.php`.

```
Artisan::add(new MyCustomCommand);
```

Believe it or not; that's it! You may now call your custom command from the terminal.

20. Mock Facades

Laravel leverages the facade pattern heavily. This allows for the clean static-like syntax that you'll undoubtedly come to love (`Route::get()`, `Config::get()`, etc.), while still allowing for complete testability behind the scenes.

Because these "underlying classes" are resolved out of the IoC container, we can easily swap those underlying instances out with mocks, for the purposes of testing. This allows for such control as:

```
Validator::shouldReceive('make')->once();
```

Yep, we're calling `shouldReceive` directly off of the facade. Behind the scenes, Laravel makes use of the excellent Mockery framework to allow for this. This means that you may freely use these facades in your code, while still allowing for 100% testability.

21. Form Helpers

Because building forms can frequently be a cumbersome task, Laravel's form builder steps in to ease the process, as well as leverage many of the idiosyncrasies related to form construction. Here are a few examples:

```
{{ Form::open() }}
```

```
    {{ Form::text('name') }}
    {{ Form::textarea('bio') }}
    {{ Form::selectYear('dob', date('Y') - 80, date('Y')) }}
{{ Form::close() }}
```

What about tasks, such as remembering input from the previous form submission? Laravel can do all of that automatically!

22. The IoC Container

At the core of Laravel is its powerful IoC container, which is a tool that assists in managing class dependencies. Notably, the container has the power to automatically resolve classes without configuration!

Simply typehint your dependencies within the constructor, and, upon instantiation, Laravel will use PHP's Reflection API to intelligently read the typehints, and attempt to inject them for you.

```
public function __construct(MyDependency $thing)
{
    $this->thing = $thing;
}
```

As long as you request the class out of the IoC container, this resolution will take place automatically.

```
$myClass = App::make('MyClass');
```

> *An important note is that controllers are always resolved out of the IoC container. As such, you may free typhint your controller's dependencies, and Laravel will subsequently do its best to inject them for you.*

23. Environments

While a single environment might work for small projects, for any applications of size, multiple environments will prove essentials. Development, testing, production...all of these are essential and require their own configuration.

Maybe your test environment uses a database in memory for testing. Maybe your development environment uses different API keys. Likely, your production environment will use a custom database connection string.

Luckily, Laravel makes our job simple once again. Have a look at `bootstrap/start.php` in your app.

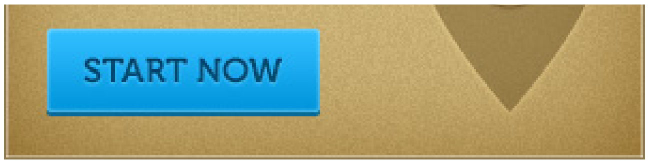
Here's a basic demonstration of setting both a `local` and `production` environment, based upon the browser's address bar.

```php
$env = $app->detectEnvironment(array(
    'local' => array('localhost'),
    'production' => array('*.com')
));
```

While this will work, generally speaking, it's preferred to use environment variables for this sort of thing. No worries; this is still doable in Laravel! Instead, simply return a function from the `detectEnvironment` method on the container object.

```php
$env = $app->detectEnvironment(function()
{
    return getenv('ENV_NAME') ?: 'local';
});
```

Now, unless an environment variable has been set (which you will do for production), the environment will default to `local`.



Advertisement

24. Simple Configuration

Laravel, yet again, takes a dead-simple approach to configuration. Create a folder within `app/config` that matches your desired environment, and any configuration files within it will take precedence, if the environment name matches. As such, to, say, set a different billing API key for development, you could do:

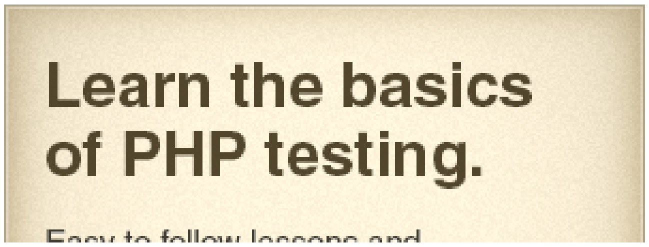
```
<?php app/config/development/billing.php

return [
    'api_key' => 'your-development-mode-api-key'
];
```

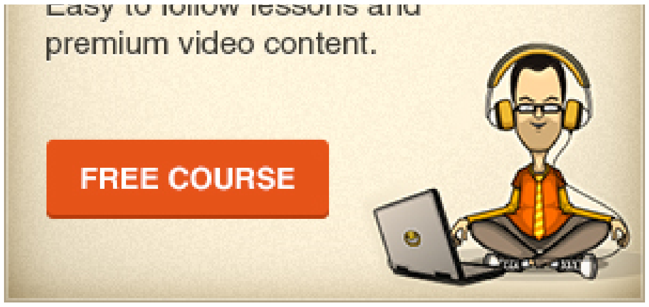
The configuration switcharoo is automatic. Simply type `Config::get('billing.api_key')`, and Laravel will correctly determine which file to read from.

25. Top-Shelf Further Learning

When it comes to education, the Laravel community, despite its relatively young age, is a never-ending well. In little over a year, a half-dozen different books - related to all matters of Laravel development - have been released.

Learn everything from the basics, to testing, to building and maintaining large apps!





Advertisement

Difficulty:

Intermediate

Length:

Medium

Categories:

Laravel **PHP**

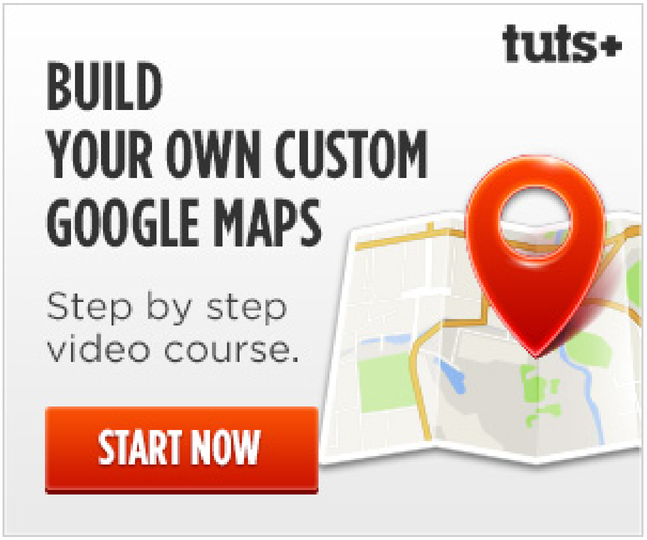
Translations:

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

Translate this post Powered by native

About Jeffrey Way

I used to be the editor of Nettuts+ and head of web development courses at Tuts+.



Advertisement

Suggested Envato Tuts+ Course

Build a CMS With Laravel $9

Related Tutorials

Doctrine ORM and Laravel 5

Code

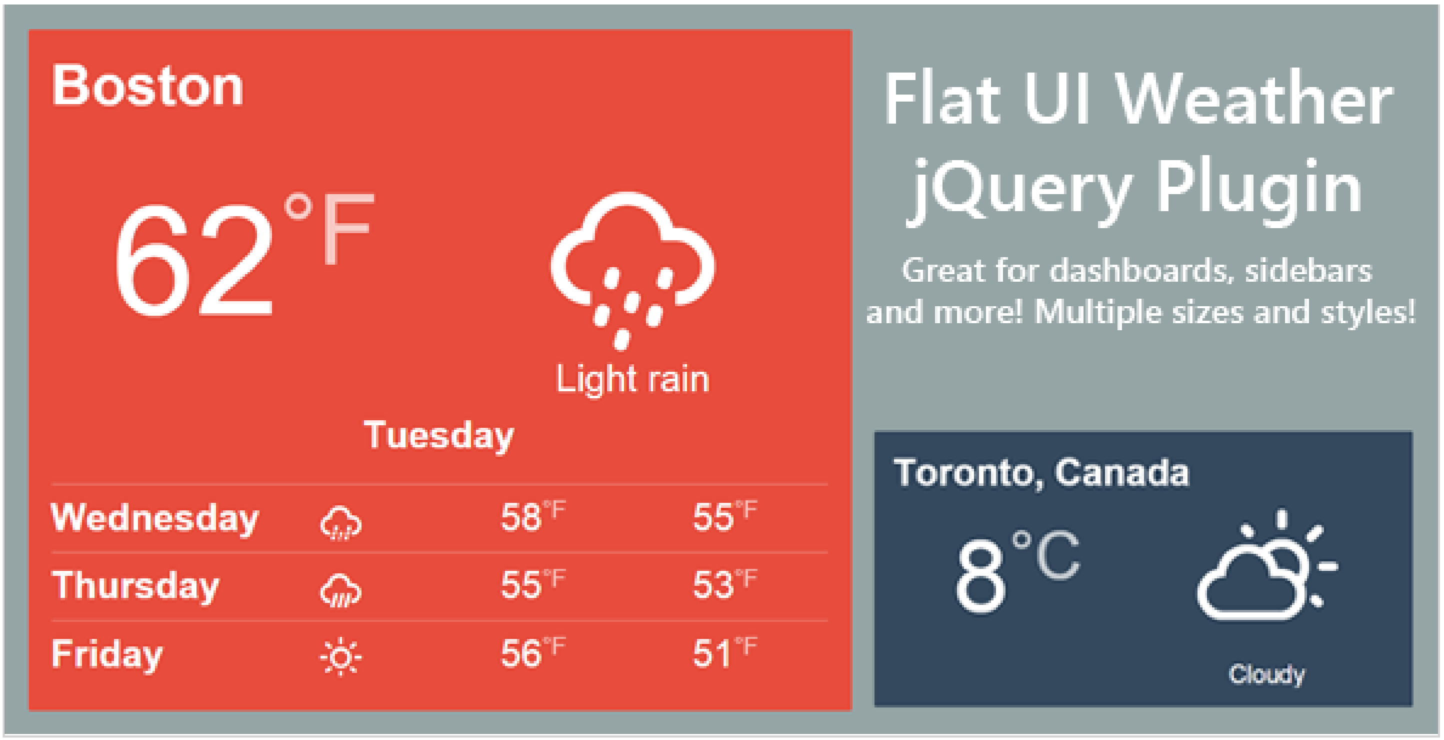
Build a Real-Time Chat Application With Modulus and Laravel 5

Code

Using Laravel 5's Authentication Facade

Code

Envato Market Item



What Would You Like to Learn?

Suggest an idea to the content editorial team at Envato Tuts+.

Email Newsletters

Get Envato Tuts+ updates, news, surveys & offers.

Email Address

Subscribe

Privacy Policy

Custom digital services like logo design, WordPress installation, video production and more.

Check out Envato Studio

Build anything from social networks to file upload systems. Build faster with pre-coded PHP scripts.

Browse PHP on CodeCanyon