Jan 27th, 2015

Introducing the Laravel 5 Command Scheduler

Suppose you wanted to create a new TODOParrot revenue stream by adding a productivity-centric book catalog to the site. Interested readers would click through to Amazon, and you would earn money on any purchases via your Amazon Associates Account (https://affiliate-program.amazon.com/). Of course, in an effort to convert as many sales as possible you'll want to ensure your book catalog always contains the latest available book covers, descriptions, and prices, something you'd rather not do manually.

Fortunately, you can automate such updates using the Amazon Product Advertising API (https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html). To implement such a solution you would typically write a script using a package such as ApaiIO (https://github.com/Exeu/apai-io), and then schedule the script's execution using your server's Cron (http://en.wikipedia.org/wiki/Cron) service. While this approach certainly works, managing task scheduling outside of your application code is pretty inconvenient.

Laravel 5 removes this inconvenience with the introduction of a command scheduler. The Laravel command scheduler allows you to manage your task execution dates and times using easily understandable PHP syntax. You'll manage the task execution definitions in `app/Console/Kernel.php`, which is presented below. You'll see that an example task has already been defined in the `schedule` method to run every hour:

```
<?php namespace todoparrot\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel {

    protected $commands = [
        'todoparrot\Console\Commands\Inspire',
    ];

    protected function schedule(Schedule $schedule)
    {
        $schedule->command('inspire')->hourly();
    }

}
```

The protected `$commands` property registers any custom commands you'd like to include in the Artisan `list` output. An example custom command (`Inspire`) is already defined, which you'll find in `app/Console/Commands/Inspire.php` . Whether you plan on scheduling custom Artisan commands or executing them directly from the terminal you'll need to reference You're not strictly limited to scheduling Artisan commands, although as you'll soon see it is quite easy to create custom Artisan commands containing the desired logic.

The `inspire` command registered in the `$commands` array is scheduled for execution in the `schedule` method. In this example you can see it has been scheduled to execute every hour (at the top of the hour). I'll talk about other scheduling options in a moment. If you execute the `inspire` command manually you'll be presented with a random quote:

```
$ php artisan inspire
Simplicity is the ultimate sophistication. - Leonardo da Vinci
```

Next I'll show you how you can create your own custom Artisan command and schedule it for execution using the command scheduler.

Creating a Custom Artisan Command

You can create your own Artisan commands which can neatly package any PHP logic you desire. To create a command use the `make:console` generator:

```
$ php artisan make:console UpdateCatalog --command=amazon:update
Console command created successfully.
```

This creates a command skeleton in `app/Console/Commands/UpdateCatalog.php` . For organizational purposes I've define a custom command name `amazon:update` , as perhaps in the future I'd like to create other Amazon-related commands and so would like them all placed under the `amazon` namespace. Open up `app/Console/Commands/UpdateCatalog.php` and you'll find the following class:

```
<?php namespace todoparrot\Console\Commands;

use Illuminate\Console\Command;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Input\InputArgument;

class UpdateCatalog extends Command {

  protected $name = 'amazon:update';

  protected $description = 'Command description.';

  public function __construct()
  {
    parent::__construct();
  }

  public function fire()
  {
    //
  }

  protected function getArguments()
  {
    return [
      ['example', InputArgument::REQUIRED,
        'An example argument.'],
    ];
  }

  protected function getOptions()
  {
    return [
      ['example', null, InputOption::VALUE_OPTIONAL,
        'An example option.', null],
    ];
  }

}
```

The `$name` and `$description` properties define the command's execution name and description, respectively, both of which will be included in the Artisan `list` output once we register it within the `app/Console/Kernel.php` `$commands` array. The `fire` method encapsulates the logic which will execute when the command is run. The `getArguments` and `getOptions` methods can be used to define both required and optional command arguments and options, respectively.

You'll see that the `getArguments` method defines a required argument. For the purposes of this exercise we're not interested in arguments nor options, so comment out the `return` statement:

```
protected function getArguments()
{
return [
  // ['example', InputArgument::REQUIRED, 'An example argument.'],
];
}
```

The Laravel documentation discusses Artisan command arguments, options, and other features. See this page (http://laravel.com/docs/master/commands) for more information.

Next, update the `fire` method to look like this:

```
  public function fire()
  {
    $this->info("Amazon catalog updated!");
  }
```

Save your changes and then register the command within `app/Console/Kernel.php` :

```
protected $commands = [
    'todoparrot\Console\Commands\Inspire',
    'todoparrot\Console\Commands\UpdateCatalog'
];
```

After saving the changes you should see the custom command in the Artisan `list` output:

```
$ php artisan list
...
Available commands:
...
amazon
 amazon:update        Updates the TODOParrot book catalog.
...
```

You can now execute the `amazon:update` command from the terminal:

```
$ php artisan amazon:update
Amazon catalog updated!
```

Scheduling Your Command

As was perhaps made obvious by the earlier example, scheduling your command within `app/Console/Kernel.php` is easy. If you'd like `amazon:update` to run hourly, you'll use the `hourly` method:

```
protected function schedule(Schedule $schedule)
{
    $schedule->command('amazon:update')->hourly();
}
```

Updating Amazon product information hourly seems a bit aggressive. Fortunately, you have plenty of other options. To run a command on a daily basis (midnight), use `daily` :

```
$schedule->command('amazon:update')->daily();
```

To run it at a specific time, use the `dailyAt` method:

```
$schedule->command('amazon:update')->dailyAt('18:00');
```

If you need to run a command very frequently, you can use an `every` method:

```
$schedule->command('amazon:update')->everyFiveMinutes();
$schedule->command('amazon:update')->everyTwentyMinutes();
```

See the Laravel documentation (http://laravel.com/docs/master/artisan#scheduling-artisan-commands) for other scheduling options.

Enabling the Scheduler

With your tasks created and scheduled, you'll need to add a single entry to your server's `crontab` file:

```
* * * * * php /path/to/artisan schedule:run 1>> /dev/null 2>&1
```

Once saved, your application's `schedule:run` Artisan command will run once per minute. It will in turn execute any jobs that you've defined using the Laravel command scheduler.

Other Scheduling Options

If defining a custom Artisan command seems overkill, you can optionally define some logic for execution directly within the `schedule` method:

```
$schedule->call(function()
{
    // Send some e-mail

})->daily();
```

You can also schedule terminal commands for execution like so:

```
$schedule->exec('/path/to/some/command')->daily();
```

Conclusion

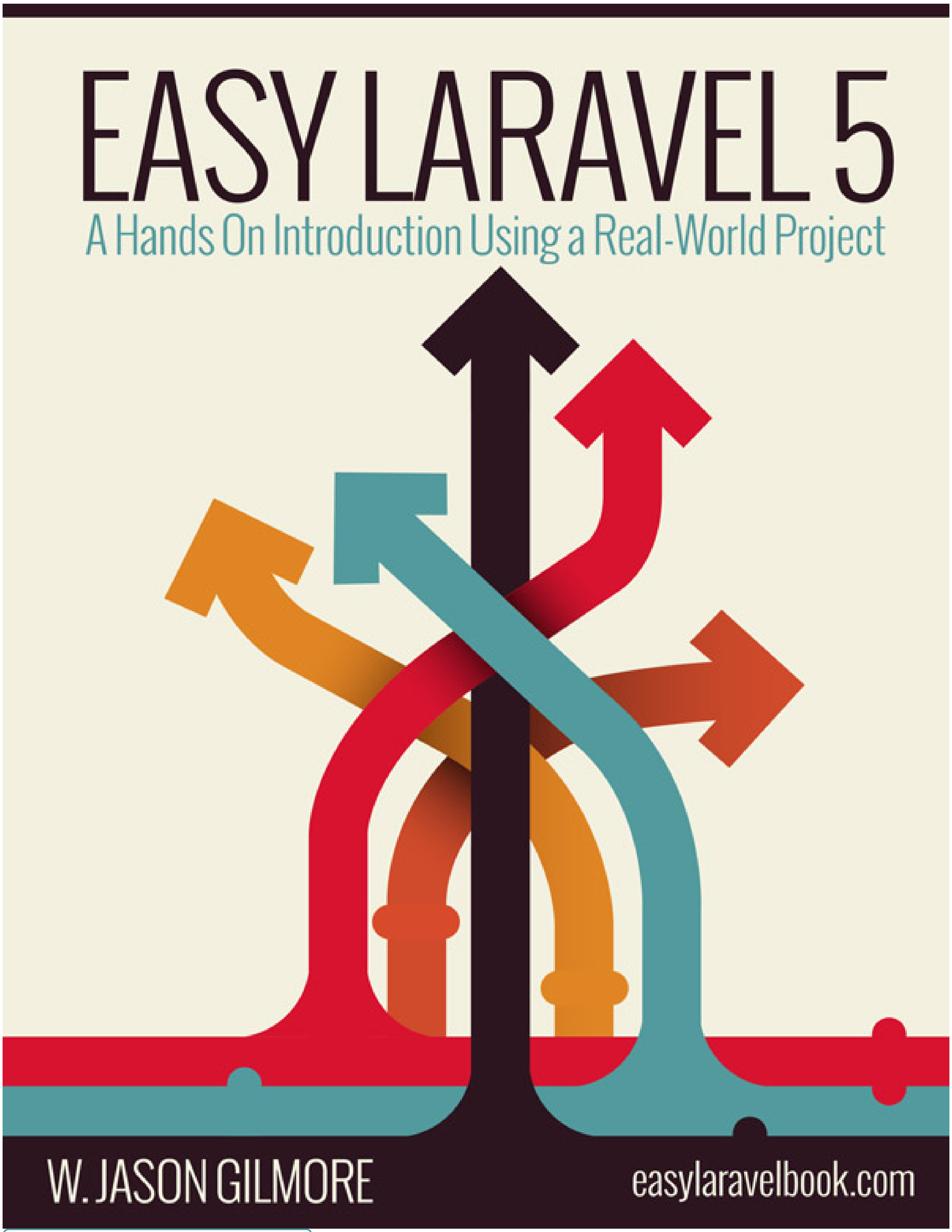
The new Laravel 5 command scheduler removes the hassle of having to separately manage another moving part to your application, allowing you to conveniently manage scheduling directly within your application's code base. If you're doing something cool with this new feature tell us about it in the comments!

Posted by W. Jason Gilmore Jan 27th, 2015

Tweet

« Creating Polymorphic Relations in Laravel 5 (/blog/2015/01/21/creating-polymorphic-relations-in-laravel-5/)

Deploying a Laravel Application to Heroku » (/blog/2015/01/31/deploying-a-laravel-application-to-heroku/)



Buy the Book (/purchase)

LaraBrain (http://larabrain.com)

The Latest LaraBrain Tips

- Automatic pluralization with str_plural() helper (http://larabrain.com/tips/automatic-pluralization-with-str-plural-helper)

- Formatting Timestamps in Laravel (http://larabrain.com/tips/formatting-timestamps-in-laravel)
- Dropping Multiple Columns in a Single Line of Laravel Migration (http://larabrain.com/tips/dropping-multiple-columns-in-a-single-line-of-laravel-migration)
- Setting a Laravel Select Box Default Value (http://larabrain.com/tips/setting-a-laravel-select-box-default-value)
- Viewing the Response Within a Laravel PHPUnit Test (http://larabrain.com/tips/viewing-the-response-within-a-laravel-phpunit-test)

Subscribe to receive a 37 page sample chapter, and occasional e-mails about Laravel and the book. No spam ever.

Your e-mail address

Subscribe

Recent Posts

Adding Custom Fields to a Laravel 5 Registration Form (/blog/2015/09/25/adding-custom-fields-to-a-laravel-5-registration-form/)

Paginating Laravel Database Results With Custom Scopes (/blog/2015/09/24/paginating-laravel-database-results-with-custom_scopes/)

Logging an Array in Laravel (/blog/2015/09/04/logging-an-array-in-laravel/)

Passing a Parameter Into Laravel Form::open (/blog/2015/08/26/passing-a-parameter-into-laravel-form-open/)

Populating a Laravel Form Multiple Select Box With hasMany Relation Values (/blog/2015/08/25/populating-a-laravel-form-multiple-select-box-with-hasmany-relation-values/)