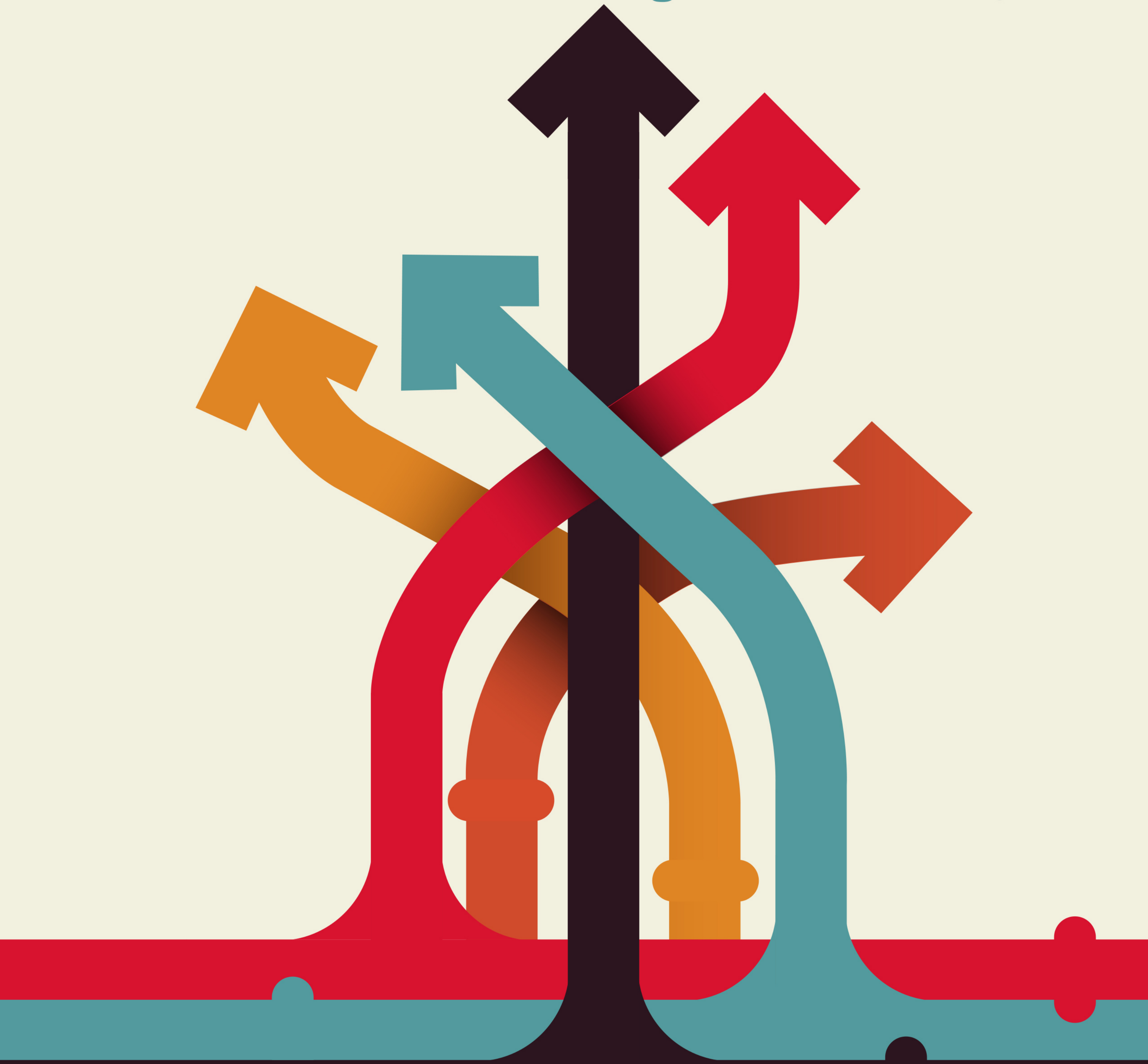
EASY LARAVEL 5

A Hands On Introduction Using a Real-World Project



W. JASON GILMORE

easylaravelbook.com

Easy Laravel 5

A Hands On Introduction Using a Real-World Project

W. Jason Gilmore

This book is for sale at http://leanpub.com/easylaravel

This version was published on 2015-10-20

Also By W. Jason Gilmore

Dedicated to The Champ, The Princess, and Little Winnie. Love, Daddy

Contents

Introduction

I've spent the vast majority of the past 15 years immersed in the PHP language. During this time I've written seven PHP-related books, including a bestseller that has been in print for more than ten years. Along the way I've worked on dozens of PHP-driven applications for clients ranging from unknown startups to globally-recognized companies, penned hundreds of articles about PHP and web development for some of the world's most popular print and online publications, and instructed hundreds of developers in the United States and Europe. So you might be surprised to learn that a few years ago I became rather disenchanted with PHP. It felt like there were more exciting developments taking place within other programming communities, and wanting to be part of that buzz, I wandered off. In recent years, I spent the majority of my time working on a variety of projects including among others several ambitious Ruby on Rails applications and even a pretty amazing Linux-powered robotic device.

Of course, even during this time in the wilderness I kept tabs on the PHP community, watching with great interest as numerous talented developers worked tirelessly to inject that missing enthusiasm back into the language. Nils Adermann and Jordi Boggiano released the Composer[1] dependency manager. The Framework Interoperability Group[2] was formed. And in 2012 the incredibly talented Taylor Otwell[3] created the Laravel framework[4] which out of nowhere became the most popular PHP project on GitHub, quickly surpassing projects and frameworks that had been actively developed for years.

At some point I spent some time with Laravel and after a scant 30 minutes knew it was the real deal. Despite being the latest in a string of high profile PHP frameworks, Laravel is incredibly polished, offering a shallow learning curve, convenient PHPUnit integration, a great object-relational mapping solution called Eloquent, and a wide variety of other great features. The reasoning behind this pragmatic approach is laid bare in the project documentation[5], in which the Laravel development team describes their project goals:

> Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Now that's something to get excited about! In the pages to follow I promise to add you to the ranks of fervent Laravel users by providing a wide-ranging and practical introduction to its many features.

[1]https://getcomposer.org/
[2]http://www.php-fig.org/
[3]http://taylorotwell.com/
[4]http://laravel.com/
[5]http://laravel.com/docs/master

What's New in Laravel 5?

Laravel 5 is an ambitious step forward for the popular framework, offering quite a few new features. In addition to providing newcomers with a comprehensive overview of Laravel's fundamental capabilities, I'll devote special coverage to several of these new features, including:

- New Project Structure: Laravel 5 projects boast a revamped project structure. In Chapter 1 I'll review every file and directory comprising the new structure so you know exactly where to find and place project files and other assets..
- Improved Environment Configuration: Laravel 5 adopts the PHP dotenv[6] package for environment configuration management. I think Laravel 4 users will really find the new approach to be quite convenient and refreshing. I'll introduce you to this new approach in Chapter 1.
- Route Annotations: The `routes.php` file remains in place for Laravel 5, however users now have the choice of alternatively using route annotations for route definitions. I'll show you how to use route annotations in Chapter 2.
- Elixir: Elixir[7] offers Laravel users a convenient way to automate various development tasks using Gulp[8], among them CSS and JavaScript compilation, JavaScript linting, image compression, and test execution. I'll introduce you to Elixir in Chapter 2.
- Flysystem: Laravel 5 integrates Flysystem[9], which allows you to easily integrate your application with remote file systems such as Dropbox, S3 and Rackspace.
- Form Requests: Laravel 5's new form requests feature greatly reduces the amount of code you'd otherwise have to include in your controller actions when validating and processing form data. In Chapter 5 I'll introduce you to this great new feature.
- Middleware: Laravel 5 introduces easy middleware integration. Middleware is useful when you want to interact with your application's request and response process in a way that doesn't pollute your application-specific logic. Chapter 7 is devoted entirely to this topic.
- Easy User Authentication: User account integration is the norm these days, however integrating user registration, login, logout, and password recovery into an application is often tedious and time-consuming. Laravel 5 all but removes this hassle by offering these features as a turnkey solution. I'll introduce you to these exciting capabilities in Chapter 6.

About this Book

This book is broken into eleven chapters and two appendices, each of which is briefly described below.

[6]https://github.com/vlucas/phpdotenv
[7]https://github.com/laravel/elixir
[8]http://gulpjs.com/
[9]https://github.com/thephpleague/flysystem

Chapter 1. Introducing Laravel

In this opening chapter you'll learn how to create and configure your Laravel project both using your existing PHP development environment and Laravel Homestead. I'll also show you how to properly configure your environment for effective Laravel debugging, and how to expand Laravel's capabilities by installing several third-party Laravel packages that promise to supercharge your development productivity. We'll conclude the chapter with an introduction to PHPUnit, showing you how to create and execute your first Laravel unit test!

Chapter 2. Managing Your Project Controllers, Layout, Views, and Other Assets

In this chapter you'll learn how to create controllers and actions, and define the routes used to access your application endpoints using Laravel 5's new route annotations feature. You'll also learn how to create the pages (views), work with variable data and logic using the Blade templating engine, and reduce redundancy using layouts and view helpers. I'll also introduce Laravel Elixir, a new feature for managing Gulp[10] tasks, and show you how to integrate the popular Bootstrap front-end framework and jQuery JavaScript library. We'll conclude the chapter with several examples demonstrating how to test your controllers and views using PHPUnit.

Chapter 3. Talking to the Database

In this chapter we'll turn our attention to the project's data. You'll learn how to integrate and configure the database, create and manage models, and interact with the database through your project models. You'll also learn how to deftly configure and traverse model relations, allowing you to greatly reduce the amount of SQL you'd otherwise have to write to integrate a normalized database into your application.

Chapter 4. Model Relations, Scopes, and Other Advanced Features

Building and navigating table relations is an standard part of the development process even when working on the most unambitious of projects, yet this task is often painful when working with many web frameworks. Fortunately, using Laravel it's easy to define and traverse these relations. In this chapter I'll show you how to define, manage, and interact with one-to-one, one-to-many, many-to-many, has many through, and polymorphic relations. You'll also learn about a great feature known as scopes which encapsulate the logic used for more advanced queries, thereby hiding it from your controllers.

[10]http://gulpjs.com/

Chapter 5. Integrating Web Forms

Your application will almost certainly contain at least a few web forms, which will likely interact with the models, meaning you'll require a solid grasp on Laravel's form generation and processing capabilities. While creating simple forms is fairly straightforward, things can complicated fast when implementing more ambitious solutions such as forms involving multiple models. In this chapter I'll go into extensive detail regarding how you can integrate forms into your Laravel applications, introducing Laravel 5's new form requests feature, covering both Laravel's native form generation solutions as well as several approaches offered by popular packages. You'll also learn how to upload files using a web form and Laravel's fantastic file upload capabilities.

Chapter 6. Integrating Middleware

Laravel 5 introduces middleware integration. In this chapter I'll introduce you to the concept of middleware and the various middleware solutions bundled into Laravel 5. You'll also learn how to create your own middleware solution!

Chapter 7. Authenticating and Managing Your Users

Most modern applications offer user registration and preference management features in order to provide customized, persisted content and settings. In this chapter you'll learn how to integrate user registration, login, and account management capabilities into your Laravel application.

Chapter 8. Deploying, Optimizing and Maintaining Your Application

"Deploy early and deploy often" is an oft-quoted mantra of successful software teams. To do so you'll need to integrate a painless and repeatable deployment process, and formally define and schedule various maintenance-related processes in order to ensure your application is running in top form. In this chapter I'll introduce the Laravel 5 Command Scheduler, which you can use to easily schedule rigorously repeating tasks. I'll also talk about optimization, demonstrating how to create a faster class router and how to cache your application routes. Finally, I'll demonstrate just how easy it can be to deploy your Laravel application to the popular hosting service Heroku, and introduce Laravel Forge.

Chapter 9. Creating a Restricted Administration Console

This chapter shows you how to identify certain users as administrators and then grant them access to a restricted web-based administrative console using a prefixed route grouping and custom middleware.

Chapter 10. Introducing the Lumen Microframework

This chapter introduces the new Laravel Lumen microframework. You'll learn all about Lumen fundamentals while building a companion microservice for the TODOParrot companion application!

Chapter 11. Introducing Events

This chapter introduces Laravel Events, showing you how to create event handlers, event listeners, and integrate events into your application logic. You'll also learn all about Laravel 5.1's fascinating event broadcasting capabilities, accompanied by a real-world example.

Appendix A. Deploying Your Laravel Application to DreamHost

In this appendix you'll learn how to configure Capistrano and GitHub to effortlessly deploy a Laravel project to the popular hosting provider DreamHost. Even if you're using another shared hosting provider, much of what is discussed in this appendix will apply to your own specific circumstances.

Appendix B. Feature Implementation Cheat Sheets

The book provides occasionally exhaustive explanations pertaining to the implementation of key Laravel features such as controllers, migrations, models and views. However, once you understand the fundamentals it isn't really practical to repeatedly reread parts of the book just to for instance recall how to create a model with a corresponding migration or seed the database. So I thought it might be useful to provide an appendix which offered a succinct overview of the steps necessary to carry out key tasks. This is a work in progress, but already contains several pages of succinct explanations.

Introducing the TODOParrot Project

Learning about a new technology is much more fun and practical when introduced in conjunction with real-world examples. Throughout this book I'll introduce Laravel concepts and syntax using code found in TODOParrot[11], a web-based task list application built atop Laravel.

The TODOParrot code is available on GitHub at https://github.com/wjgilmore/todoparrot[12]. It's released under the MIT license, so feel free to download the project and use it as an additional learning reference or in any other manner adherent to the licensing terms.

[11]http://todoparrot.com

[12]https://github.com/wjgilmore/todoparrot

About the Author

W. Jason Gilmore[13] is a software developer, consultant, and bestselling author. He has spent much of the past 15 years helping companies of all sizes build amazing solutions. Recent projects include a Rails-driven e-commerce analytics application for a globally recognized publisher, a Linux-powered autonomous environmental monitoring buoy, and a 10,000+ product online store.

Jason is the author of seven books, including the bestselling "Beginning PHP and MySQL, Fourth Edition", "Easy Active Record for Rails Developers", and "Easy PHP Websites with the Zend Framework, Second Edition".

Over the years Jason has published more than 300 articles within popular publications such as Developer.com, JSMag, and Linux Magazine, and instructed hundreds of students in the United States and Europe. Jason is cofounder of the wildly popular CodeMash Conference[14], the largest multi-day developer event in the Midwest.

Away from the keyboard, you'll often find Jason playing with his kids, hunched over a chess board, and having fun with DIY electronics.

Jason loves talking to readers and invites you to e-mail him at wj@wjgilmore.com.

Errata and Suggestions

Nobody is perfect, particularly when it comes to writing about technology. I've surely made some mistakes in both code and grammar, and probably completely botched more than a few examples and explanations. If you would like to report an error, ask a question or offer a suggestion, please e-mail me at wj@wjgilmore.com.

[13]http://www.wjgilmore.com
[14]http://www.codemash.org

Chapter 1. Introducing Laravel

Laravel is a web application framework that borrows from the very best features of other popular framework solutions, among them Ruby on Rails and ASP.NET MVC. For this reason, if you have any experience working with other frameworks then I'd imagine you'll make a pretty graceful transition to Laravel-driven development. If this is your first acquaintance with framework-driven development, you're in for quite a treat! Frameworks are so popular precisely because they dramatically decrease the amount of work you'd otherwise have to do by making many of the mundane decisions for you, a concept known as convention over configuration[15].

In this chapter you'll learn how to install Laravel and create your first Laravel project. We'll use this project as the basis for introducing new concepts throughout the remainder of the book, and to keep things interesting I'll base many of the examples around the TODOParrot application introduced in this book's introduction. I'll also introduce you to several powerful debugging and development tools that I consider crucial to Laravel development, showing you how to integrate them into your development environment. Finally, I'll show you how to configure Laravel's testing environment in order to create powerful automated tests capable of ensuring your Laravel application is operating precisely as expected.

> I published this book on February 4, 2015, the very same day Laravel 5 officially released. Since then I've made more than one hundred improvements and expansions, and often fix any reported errata within a few days following notification (see http://easylaravelbook.com/changelog/). More recently this includes a *major* book revision to reflect Laravel 5.1 changes. If you find an issue please e-mail me at wj@wjgilmore.com.

Installing Laravel

Laravel is a PHP-based framework that you'll typically use in conjunction with a database such as MySQL or PostgreSQL. Therefore, before you can begin building a Laravel-driven web application you'll need to first install PHP 5.4 or newer and one of Laravel's supported databases (MySQL, PostgreSQL, SQLite, and Microsoft SQL Server). Therefore if you're already developing PHP-driven web sites and are running PHP 5.4 then installing Laravel will be a breeze, and you can jump ahead to the section "Creating the TODOParrot Application". If this is your first encounter with PHP then please take some time to install a PHP development environment now. How this is accomplished depends upon your operating system and is out of the scope of this book, however there are plenty

[15] http://en.wikipedia.org/wiki/Convention_over_configuration

of available online resources. If you have problems finding a tutorial suitable to your needs, please e-mail me and I'll help you find one.

Alternatively, if you'd rather go without installing a PHP development environment at this time, you have a fantastic alternative at your disposal called *Homestead*.

Laravel currently supports several databases, including MySQL, PostgreSQL, SQLite, and Microsoft SQL Server.

Introducing Homestead

PHP is only one of several technologies you'll need to have access to in order to begin building Laravel-driven web sites. Additionally you'll need to install a web server such as Apache[16] or nginx[17], a database server such as MySQL[18] or PostgreSQL[19], and often a variety of supplemental technologies such as Redis[20] and Grunt[21]. As you might imagine, it can be quite a challenge to install and configure all of these components, particularly when you'd prefer to be writing code instead of grappling with configuration issues.

In recent years the bar was dramatically lowered with the advent of the *virtual machine*. A virtual machine is a software-based implementation of a computer that can be run inside the confines of another computer (such as your laptop), or even inside another virtual machine. This is an incredibly useful bit of technology, because you can use a virtual machine to for instance run Ubuntu Linux inside Windows 7, or vice versa. Further, it's possible to create a customized virtual machine image preloaded with a select set of software. This image can then be distributed to fellow developers, who can run the virtual machine and take advantage of the custom software configuration. This is precisely what the Laravel developers have done with Homestead[22], a Vagrant[23]-based virtual machine which bundles everything you need to get started building Laravel-driven websites.

Homestead is currently based on Ubuntu 14.04, and includes everything you need to get started building Laravel applications, including PHP 5.6, Nginx, MySQL, PostgreSQL and a variety of other useful utilities. It runs flawlessly on OS X, Linux and Windows, and Vagrant configuration is pretty straightforward, meaning in most cases you'll have everything you need to begin working with Laravel in less than 30 minutes.

[16]http://httpd.apache.org/
[17]http://nginx.org/
[18]http://www.mysql.com/
[19]http://www.postgresql.org/
[20]http://redis.io/
[21]http://gruntjs.com/
[22]http://laravel.com/docs/homestead
[23]http://www.vagrantup.com/

Installing Homestead

Homestead requires Vagrant[24] and VirtualBox[25]. User-friendly installers are available for all of the common operating systems, including OS X, Linux and Windows. Take a moment now to install Vagrant and VirtualBox. Once complete, open a terminal window and execute the following command:

```
$ vagrant box add laravel/homestead
==> box: Loading metadata for box 'laravel/homestead'
    box: URL: https://vagrantcloud.com/laravel/homestead
==> box: Adding box 'laravel/homestead' (v0.2.2) for provider: virtualbox
    box: Downloading: https://vagrantcloud.com/laravel/boxes/homestead/
    versions/0.2.2/providers/virtualbox.box
==> box: Successfully added box 'laravel/homestead' (v0.2.2) for 'virtualbox'!
```

Throughout the book I'll use the $ to symbolize the terminal prompt.

This command installs the Homestead *box*. A box is just a term used to refer to a Vagrant package. Packages are the virtual machine images that contain the operating system and various programs. The Vagrant community maintains a variety of boxes useful for different applications, so check out this list of popular boxes[26] for an idea of what else is available.

Once the box has been added, you'll next want to install the Homestead CLI tool. To do so, you'll use Composer:

```
$ composer global require "laravel/homestead=~2.0"
Changed current directory to /Users/wjgilmore/.composer
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
  - Installing symfony/process (v2.6.3)
    Downloading: 100%

  - Installing laravel/homestead (v2.0.8)
    Downloading: 100%

Writing lock file
Generating autoload files
```

[24]http://www.vagrantup.com/
[25]https://www.virtualbox.org/wiki/Downloads
[26]https://vagrantcloud.com/discover/popular

After this command has completed, make sure your ∼/.composer/vendor/bin directory is available within your system PATH. This is because the `laravel/homestead` package includes a command-line utility (named `homestead`) which you'll use to create your Homestead configuration directory:

```
$ homestead init
Creating Homestead.yaml file... ok
Homestead.yaml file created at: /Users/wjgilmore/.homestead/Homestead.yaml
```

Next you'll want to configure the project directory that you'll share with the virtual machine. Doing so requires you to identify the location of your public SSH key, because key-based encryption is used to securely share this directory. If you don't already have an SSH key and are running Windows, this SiteGround tutorial[27] offers a succinct set of steps. If you're running Linux or OS X, nixCraft[28] offers a solid tutorial.

You'll need to identify the location of your public SSH key in the `.homestead` directory's `Homestead.yaml` file. Open this file and locate the following line:

```
authorize: ~/.ssh/id_rsa.pub
```

If you're running Linux or OS X, then you probably don't have to make any changes to this line because SSH keys are conventionally stored in a directory named `.ssh` found in your home directory. If you're running Windows then you'll need to update this line to conform to Windows' path syntax:

```
authorize: c:/Users/wjgilmore/.ssh/id_rsa.pub
```

If you're running Linux or OS X and aren't using the conventional SSH key location, or are running Windows you'll also need to modify `keys` accordingly. For instance Windows users would have to update this section to look something like this:

```
keys:
    - c:/Users/wjgilmore/.ssh/id_rsa
```

Next you'll need to modify the `Homestead.yaml` file's `folders` list to identify the location of your Laravel project (which we'll create a bit later in this chapter). The two relevant `Homestead.yaml` settings are `folders` and `sites`, which by default look like this:

[27]http://kb.siteground.com/how_to_generate_an_ssh_key_on_windows_using_putty/
[28]http://www.cyberciti.biz/faq/how-to-set-up-ssh-keys-on-linux-unix/

```
folders:
    - map: ~/Code
      to: /home/vagrant/Code

sites:
    - map: homestead.app
      to: /home/vagrant/Code/Laravel/public
```

It's this particular step that tends to confuse most Homestead beginners, so pay close attention to the following description. The `folders` object's `map` attribute identifies the location in which your Laravel project will be located. The default value is ~/`Code`, meaning Homestead expects your project to reside in a directory named `Code` found in your home directory. You're free to change this to any location you please, keeping in mind for the purposes of this introduction the directory *must* be your Laravel project's root directory (why this is important will become apparent in a moment). The `folders` object's `to` attribute identifies the location *on the virtual machine* that will mirror the contents of the directory defined by the `map` key, thereby making the contents of your local directory available to the virtual machine.

The `sites` object's `map` attribute defines the domain name used to access the Laravel application via the browser. Leave this untouched for now. Finally, the `sites` object's `to` attribute defines the Laravel project's root *web directory*, which is `/public` by default. This isn't just some contrived setting; not only is `/public` the directory you would need to configure when setting up a web server to serve a Laravel application, but `/home/vagrant/Code/Laravel/public` is also the directory that Homestead's nginx web server has been configured to use! This means that the path defined by the `folders` `map` attribute *must* contain a directory named `Laravel`, and inside that a directory named `public`. If you do not do this you'll receive the dreaded `404` error when attempting to access the application via your browser.

If this explanation is clear as mud, let's clarify with an example. Begin by setting the `folders` object's `map` attribute to any path you please, likely somewhere within the directory where you tend to manage your various software projects. For instance, mine is currently set like this:

```
folders:
    - map: /Users/wjgilmore/Software/dev.todoparrot.com
    - to: /home/vagrant/Code
```

Next, create a directory named `Laravel` inside the directory identified by the `map` attribute, and inside it create a directory named `public`. Create a file named `index.php` inside the `public` directory, adding the following contents to it:

```
<?php echo "Hello from Homestead!"; ?>
```

Save these changes, and then run the following command:

```
$ homestead up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'laravel/homestead'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'laravel/homestead' is up to date...
...
==> default: Forwarding ports...
default: 80 => 8000 (adapter 1)
default: 443 => 44300 (adapter 1)
default: 3306 => 33060 (adapter 1)
default: 5432 => 54320 (adapter 1)
default: 22 => 2222 (adapter 1)
$
```

Your Homestead virtual machine is up and running! Open a browser and navigate to the URL `http://localhost:8000` and you should see `Hello from Homestead!`. Note the use of the `8000` port in the URL. This is because the Homestead virtual machine forwards several key ports to non-standard port numbers, allowing you to continue using the standard ports locally. I've included the list of forwarded ports in the debug output that followed the `vagrant up` command. As you can see, port 80 (for HTTP) forwards to 8000, port 3306 (for MySQL) forwards to 33060, port 5432 (for PostgreSQL) forwards to 54321, and port 22 (for SSH) forwards to 2222.

Next you'll want to update your development machine's `hosts` file so you can easily access the server via a hostname rather than the IP address found in the `Homestead.yaml` file. If you're running OSX or Linux, this file is found at `/etc/hosts`. If you're running Windows, you'll find the file at `C:\Windows\System32\drivers\etc\hosts`. Open up this file and add the following line:

```
192.168.10.10  homestead.app
```

After saving the changes, open a browser and navigate to `http://homestead.app`. If the virtual machine did not start, or if you do not see `Hello from Homestead!` when accessing `http://homestead.app`, then double-check your `Homestead.yaml` file, ensuring all of the paths are properly set.

Remember, we just created the `Laravel/public` directory to confirm Homestead is properly configured and able to serve files found in our local development directory. You should subsequently delete this directory as it will be created automatically when we generate the book theme project in the section, "Creating the TODOParrot Application".

Incidentally, if you'd like to shut down the virtual machine you can do so using the following command:

```
$ homestead halt
==> default: Attempting graceful shutdown of VM...
$
```

If you'd like to competely delete the virtual machine (including all data within it), you can use the `destroy` command:

```
$ homestead destroy
```

SSH'ing Into Your Virtual Machine

Because Homestead is a virtual machine running Ubuntu, you can SSH into it just as you would any other server. For instance you might wish to configure nginx or MySQL, install additional software, or make other adjustments to the virtual machine environment. You can SSH into the virtual machine using the `ssh` command if you're running Linux or OS X, or using a variety of SSH clients if you're running Windows (My favorite Windows SSH client is PuTTY[29].:

```
$ ssh vagrant@127.0.0.1 -p 2222
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Thu Jan  8 00:57:20 UTC 2015

  System load:  0.96               Processes:           104
  Usage of /:   5.0% of 39.34GB    Users logged in:     0
  Memory usage: 28%                IP address for eth0: 10.0.2.15
  Swap usage:   0%                 IP address for eth1: 192.168.33.10

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

Last login: Fri Dec 19 15:01:15 2014 from 10.0.2.2
```

You'll be logged in as the user `vagrant`, and if you list this user's home directory contents you'll see the `Code` directory defined in the `Homestead.yaml` file:

[29] http://www.putty.org/

```
vagrant@homestead:~$ ls
Code
```

If you're new to Linux be sure to spend some time nosing around Ubuntu! This is a perfect opportunity to get familiar with the Linux operating system without any fear of doing serious damage to a server because if something happens to break you can always reinstall the virtual machine!

Creating the TODOParrot Application

With Laravel (and optionally Homestead) installed and configured, it's time to get our hands dirty! We're going to start by creating the TODOParrot application, as it will serve as the basis for much of the instructional material presented throughout this book. There are a couple of different ways in which you can do this, but one of the easiest involves installing the Laravel installer using Composer[30]:

```
$ composer global require "laravel/installer=~1.1"
```

Obviously you'll need to install Composer to use Laravel in this fashion, however you'll need it anyway to perform other tasks such as package installation. See the Composer[31] website for more information regarding installation.

After installation you'll be able to create new Laravel project skeletons using the `laravel` utility's `new` command:

```
$ laravel new dev.todoparrot.com
Crafting application...
Application ready! Build something amazing.
```

Next you'll need to create an environment-specific configuration file. This file is used as a central repository for managing your various database connection variables, API keys, and other crucial information. Laravel provides a configuration file template, so all you need to do is create a copy:

```
$ cd dev.todoparrot.com
$ cp .env.example .env
```

Finally, you'll need to generate an application key. This key is used for managing sensitive data such as cookie encryption:

[30]https://getcomposer.org
[31]https://getcomposer.org

```
$ php artisan key:generate
```

If you're using Homestead remember that the Laravel application must reside *inside* the directory identified by the `homestead.yaml folders` object's `map` attribute. Otherwise, if you're working with a local PHP environment, you can execute it wherever you'd like the project to be managed:

This command creates a new Laravel skeleton project in the directory `dev.todoparrot.com`. These contents are a combination of files and directories, each of which plays an important role in the functionality of your application so it's important for you to understand their purpose. Let's quickly review the role of each:

- `.env`: Laravel 5 uses the PHP dotenv[32] to conveniently manage environment-specific settings. You'll use `.env` file as the basis for configuring these settings. A file named `.env.example` is also included in the project root directory. This file should be used as the setting template, which fellow developers will subsequently copy over to `.env` and change to suit their own needs. I'll talk about this file and Laravel 5's solution for managing environment settings in the later section, "Configuring Your Laravel Application".
- `.gitattributes`: This file is used by Git[33] to ensure consistent settings across machines, which is particularly useful when multiple developers using a variety of operating systems are working on the same project. The lone setting found in your project's `.gitattributes` file (`text=auto`) ensures file line endings are normalized to `LF` whenever the files are checked into the repository. Plenty of other attributes are however available; Scott Chacon's book, "Pro Git"[34] includes a section ("Customizing Git - Git Attributes"[35]) with further coverage on this topic.
- `.gitignore`: This file tells Git what files and folders should not be included in the repository. You'll see the usual suspects in here, including the annoying OS X `.DS_Store` file, Windows' equally annoying `Thumbs.db` file, and the `vendor` directory, which includes the Laravel source code and various other third-party packages.
- `app`: This directory contains much of the custom code used to power your application, including the models, controllers, and middleware. We'll spend quite a bit of time inside this directory as the application development progresses.
- `artisan`: `artisan` is a command-line interface we'll use to rapidly develop new parts of your applications such as controllers, manage your database's evolution through a great feature known as *migrations*, and clear the application cache. You'll also regularly use `artisan` to interactively debug your application, and even easily view your application within the browser using the native PHP development server. We'll return to `artisan` repeatedly throughout the book as it is such an integral part of Laravel development.

[32]https://github.com/vlucas/phpdotenv
[33]http://git-scm.com/
[34]http://git-scm.com/book
[35]http://git-scm.com/book/en/Customizing-Git-Git-Attributes

- `bootstrap`: This directory contains the various files used to initialize a Laravel application, loading the configuration files, various application models and other classes, and define the locations of key directories such as `app` and `public`. Normally you won't have to modify any of the files found in the `bootstrap` directory, although I encourage you to have a look as each is heavily commented.
- `composer.json`: Composer[36] is the name of PHP's popular package manager, used by thousands of developers around the globe to quickly integrate popular third-party solutions such as Swift Mailer[37] and Doctrine[38] into a PHP application. Laravel supports Composer, and you'll use the `composer.json` file to identify the packages you'll like to integrate into your Laravel application. If you're not familiar with Composer you'll quickly come to wonder how you ever lived without it. In fact in this introductory chapter alone we'll use it several times to install several useful packages.
- `composer.lock`: This file contains information about the state of the installed Composer packages at the time these packages were last installed and/or updated. Like the `bootstrap` directory, you will rarely if ever directly interact with this file.
- `config`: This directory contains more than a dozen files used to configure various aspects of your Laravel application, such as the database credentials, and the cache, e-mail delivery and session settings.
- `database`: This directory contains the directories used to house your project's database migrations and seed data (migrations and database seeding are both introduced in Chapter 3).
- `gulpfile.js`: Laravel 5 introduces a new feature called *Laravel Elixir*. `Gulpfile.js` is used by Elixir to define various Gulp.js[39] tasks used by Elixir to automate various build-related processes associated with your project's CSS, JavaScript, tests, and other assets. I'll introduce Elixir in Chapter 2.
- `package.json`: This file is used by the aforementioned Elixir to install Elixir and its various dependencies. I'll talk about this file in Chapter 2.
- `phpspec.yml`: This file is used to configure the behavior driven development tool phpspec[40]. In this book I'll discuss Laravel testing solely in the context of PHPUnit but hope to include coverage of phpspec in a forthcoming update.
- `phpunit.xml`: Even relatively trivial web applications should be accompanied by an automated test suite. Laravel leaves little room for excuse to shirk this best practice by configuring your application to use the popular PHPUnit[41] test framework. The `phpunit.xml` is PHPUnit's application configuration file, defining characteristics such as the location of the application tests. We'll return to this topic repeatedly throughout the book, so stay tuned.
- `public`: The `public` directory serves as your application's root directory, housing the `.htaccess`, `robots.txt`, and `favicon.ico` files, in addition to a file named `index.php` that is the *first*

[36]https://getcomposer.org
[37]http://swiftmailer.org/
[38]http://www.doctrine-project.org/
[39]http://gulpjs.com/
[40]http://www.phpspec.net/
[41]http://phpunit.de/

file to execute when a user accesses your application. This file is known as the *front controller*, and it is responsible for loading and executing the application.
- `readme.md`: The `readme.md` file contains some boilerplate information about Laravel of the sort that you'll typically find in an open source project. Feel free to replace this text with information about your specific project. See the TODOParrot[42] README file for an example.
- `resources`: The `resources` directory contains your project's views and localized language files. You'll also store your project's raw assets (CoffeeScript, SCSS, etc.).
- `storage`: The `storage` directory contains your project's cache, session, and log data.
- `tests`: The `tests` directory contains your project's PHPUnit tests. Testing is a recurring theme throughout this book, complete with numerous examples.
- `vendor`: The `vendor` directory is where the Laravel framework code itself is stored, in addition to any other third-party code. You won't typically directly interact with anything found in this directory, instead doing so through the `artisan` utility and Composer interface.

Now that you have a rudimentary understanding of the various directories and files comprising a Laravel skeleton application let's see what happens when we load the default application into a browser. If you're using Homestead then navigate to `http://homestead.app`, otherwise if you plan on using PHP's built-in development server, start the server by executing the following command:

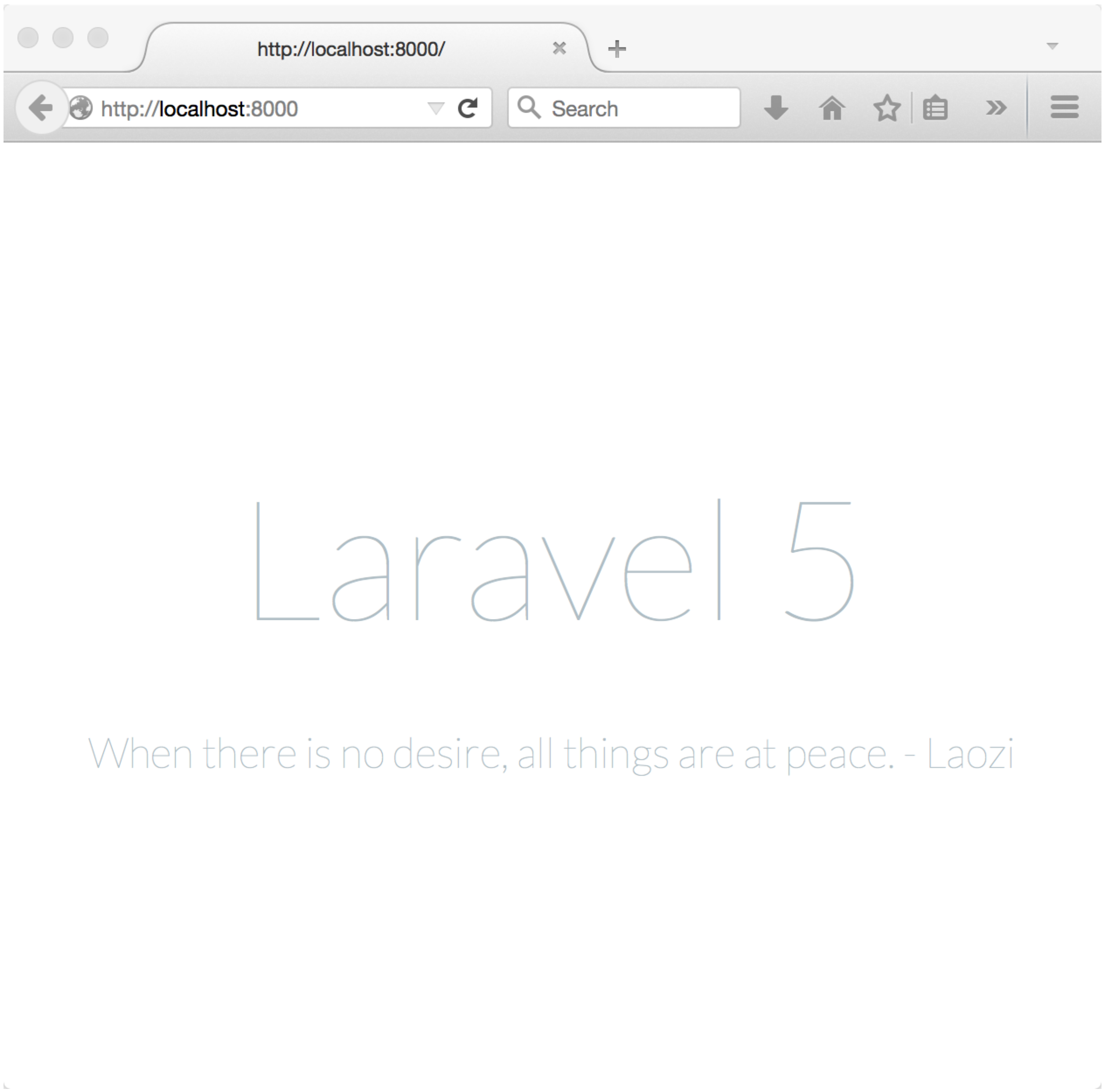
```
$ php artisan serve
Laravel development server started on http://localhost:8000
```

Alternatively you could use the built-in PHP server:

```
$ php -S localhost:8000 -t public /
PHP 5.5.15 Development Server started at Wed Jan  7 20:30:49 2015
Listening on http://localhost:8000
Document root is /Users/wjgilmore/Code/Laravel/public
Press Ctrl-C to quit.
```

Once the server is running, open your browser and navigate to the URL `http://localhost:8000`. Load this URL to your browser and you'll see the page presented in the below figure.

[42] http://github.com/wjgilmore/todoparrot



The Laravel splash page

As you can see, the Laravel logo is presented in the default page. So where is this page and logo located? It's found in a *view*, and in the next chapter I'll introduce Laravel views in great detail.

Setting the Application Namespace

Laravel 5 uses the PSR-4 autoloading standard[43], meaning your project controllers, models, and other key resources are namespaced. The default namespace is set to `app`, which is pretty generic. You'll

[43]http://www.php-fig.org/psr/psr-4/

likely want to update your project's namespace to something reasonably unique, such as `todoparrot`. You can do so using the `artisan` CLI's `app:name` command:

```
$ php artisan app:name todoparrot
Application namespace set!
```

This command will not only update the default namespace setting (by modifying `composer.json`'s `autoload/psr-4` setting), but will additionally updating any namespace declarations found in your controllers, models, and other relevant files.

Configuring Your Laravel Application

Most web frameworks, Laravel included, offer environment-specific configuration, meaning you can define certain behaviors applicable only when you are developing the application, and other behaviors when the application is running in production. For instance you'll certainly want to output errors to the browser during development but ensure errors are only output to the log in production.

Your application's default configuration settings are found in the `config` directory, and are managed in a series of files including:

- `app.php`: The `app.php` file contains settings that have application-wide impact, including whether debug mode is enabled (more on this in a moment), the application URL, timezone, locale, and autoloaded service providers.
- `auth.php`: The `auth.php` file contains settings specific to user authentication, including what model manages your application users, the database table containing the user information, and how password reminders are managed. I'll talk about Laravel's user authentication features in Chapter 7.
- `broadcasting.php`: The `broadcasting.php` is used to configure Laravel 5.1's new event broadcasting feature. I discuss event broadcasting in Chapter 11.
- `cache.php`: Laravel supports several caching drivers, including filesystem, database, memcached, redis, and others. You'll use the `cache.php` configuration file to manage various settings specific to these drivers.
- `compile.php`: Laravel can improve application performance by generating a series of files that allow for faster package autoloading. The `compile.php` configuration file allows you to define additional class files that should be included in the optimization step.
- `database.php`: The `database.php` configuration file defines a variety of database settings, including which of the supported databases the project will use, and the database authorization credentials.
- `filesystems.php`: The `filesystems.php` configuration file defines the file system your project will use to manage assets such as file uploads. Currently the local disk, Amazon S3, and Rackspace are supported.

- `mail.php`: As you'll learn in Chapter 5 it's pretty easy to send an e-mail from your Laravel application. The `mail.php` configuration file defines various settings used to send those e-mails, including the desired driver (SMTP, Sendmail, PHP's `mail()` function, Mailgun, and the Mandrill API are supported). You can also direct mails to the log file, a technique that is useful for development purposes.
- `queue.php`: Queues can improve application performance by allowing Laravel to offload time- and resource-intensive tasks to a queueing solution such as Beanstalk[44] or Amazon Simple Queue Service[45]. The `queue.php` configuration file defines the desired queue driver and other relevant settings.
- `services.php`: If your application uses a third-party service such as Stripe for payment processing or Mandrill for e-mail delivery you'll use the `services.php` configuration file to define any third-party service-specific settings.
- `session.php`: It's entirely likely your application will use sessions to aid in the management of user preferences and other customized content. Laravel supports a number of different session drivers used to facilitate the management of session data, including the file system, cookies, a database, the Alternative PHP Cache[46], Memcached, and Redis. You'll use the `session.php` configuration file to identify the desired driver, and manage other aspects of Laravel's session management capabilities.
- `view.php`: The `view.php` configuration file defines the default location of your project's view files and the renderer used for pagination.

I suggest spending a few minutes nosing around these files to get a better idea of what configuration options are available to you. There's no need to make any changes at this point, but it's always nice to know what's possible.

Programming Terminology Alert

The terms *service provider* and *facade* regularly make and appearance within Laravel documentation, tutorials and discussions. This is because Laravel was conceived with interoperability in mind, providing the utmost flexibility in terms of being able to swap out for instance one logging or authentication implementation for another, extend Laravel with a new approach to database integration, or enhance Laravel's form generation capabilities with new features. Each of these distinct features are incorporated into Laravel via a *service provider*, which is responsible for configuring the feature for use within a Laravel application. A list of service providers integrated into your newly created project can be found in the `config/app.php` file's `providers` array. Laravel users will then typically use *facades* to access the functionality made available by the classes integrated via the service providers. A facade just facilitates interaction with these classes, and nothing more. For now that's pretty much all you need to know about these two topics, but I thought at least a cursory definition of each was in order since I'll unavoidably use both terms in this chapter and beyond.

[44]http://kr.github.io/beanstalkd/
[45]http://aws.amazon.com/sqs/
[46]http://php.net/manual/en/book.apc.php

Configuring Your Environment

Laravel presumes your application is running in a production environment, meaning the options found in the various `config` files are optimized for production use. Logically you'll want to override at least a few of these options when the application is running in your development (which Laravel refers to as `local`) environment. Laravel 5 completely overhauls the approach used to detect the environment and override environment-specific settings. It now relies upon the popular PHP dotenv[47] package. You'll set the environment simply by updating the `.env` file found in your project's root directory to reflect the desired environment settings. The default `.env` file looks like this:

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=SomeRandomString

DB_HOST=localhost
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
```

Laravel will look to this file to determine which environment is being used (as defined by the `APP_ENV` variable). These variables can then be used within the configuration files via the `env` function, as demonstrated within the `config/database.php` file, which retrieves the `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD` variables:

[47] https://github.com/vlucas/phpdotenv

```
'mysql' => [
    'driver'    => 'mysql',
    'host'      => env('DB_HOST', 'localhost'),
    'database'  => env('DB_DATABASE', 'forge'),
    'username'  => env('DB_USERNAME', 'forge'),
    'password'  => env('DB_PASSWORD', ''),
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '',
    'strict'    => false,
],
```

We'll add to the configuration file as new concepts and features are introduced throughout the remainder of this book.

Useful Development and Debugging Tools

There are several native Laravel features and third-party tools that can dramatically boost productivity by reducing the amount of time and effort spent identifying and resolving bugs. In this section I'll introduce you to my favorite such solutions, and additionally show you how to install and configure the third-party tools.

The debugging and development utilities discussed in this section are specific to Laravel, and do not take into account the many other tools available to PHP in general. Be sure to check out Xdebug[48], FirePHP[49], and the many tools integrated into PHP IDEs such as Zend Studio[50] and PHPStorm[51].

The dd() Function

Ensuring the `debug` option is enabled is the easiest way to proactively view information about any application errors however it isn't a panacea for all debugging tasks. For instance, sometimes you'll want to peer into the contents of an object or array even if the data structure isn't causing any particular problem or error. You can do this using Laravel's `dd()`[52] helper function, which will dump a variable's contents to the browser and halt further script execution. If you'd like to see the `dd()` function in action we'll need to create a proper controller/action/view configuration for

48http://xdebug.org/
49http://www.firephp.org/
50http://www.zend.com/en/products/studio
51http://www.jetbrains.com/phpstorm/
52http://laravel.com/docs/helpers#miscellaneous

the home page as Laravel 5.1 includes some unfortunate changes to the default home page behavior. If you're new to concepts such as controllers, actions and views I suggest just skipping this section and returning to it after you've read chapter 2.

Begin by creating a file named `WelcomeController.php`, and save it to `app/Http/Controllers`. Add the following contents to it (I'm assuming you've changed your project namespace to `Todoparrot` otherwise you'll need to adjust that first line accordingly):

```php
<?php namespace Todoparrot\Http\Controllers;

class WelcomeController extends Controller {

    public function index()
    {
      $items = array(
        'items' => [
          'Pack luggage',
          'Go to airport',
          'Arrive in San Juan'
        ]
      );
      dd($items);
      return view('welcome');
    }

}
```

Don't get too caught up on what's going on here if you're not familiar with the concept of a controller. Controllers and these class methods (actions) will be formally introduced in Chapter 2. Next, open up app/Http/routes.php . It currently looks like this:

```php
Route::get('/', function () {
    return view('welcome');
});
```

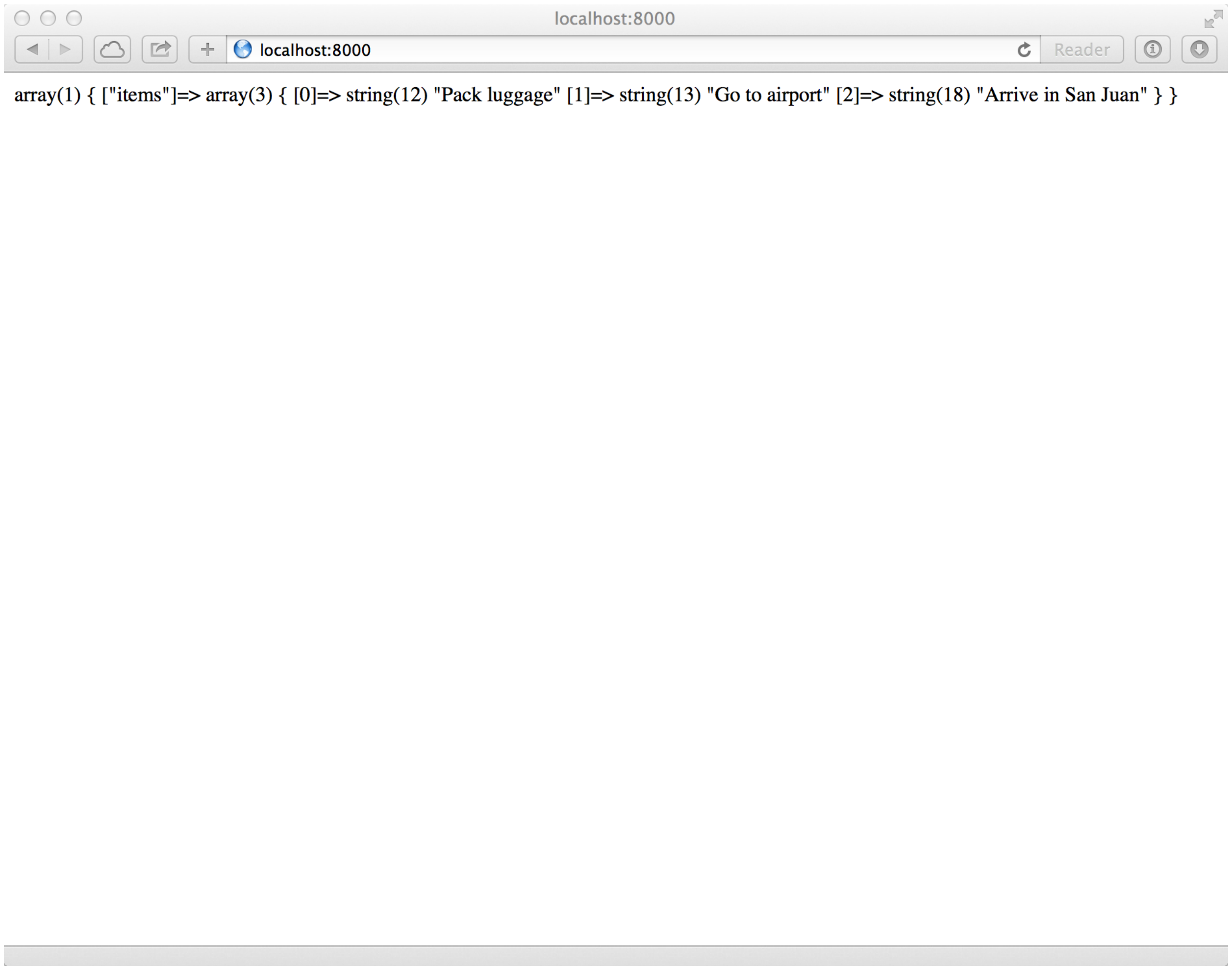
Delete that code and replace it with:

```php
Route::get('/', ['as' => 'home',
  'uses' => 'WelcomeController@index']);
```

Finally, create a file named `welcome.blade.php`, placing it inside `resources/views`. Add the following contents to it:

```
<h1>This is your new home page</h1>
```

Reload the home page in your browser and you should see the `$items` array contents dumped to the browser window as depicted in the below screenshot. Keep in mind that even though we created a display page (`welcome.blade.php`), it won't get displayed because the `dd()` function will immediately force a halt to subsequent execution and just display whatever debugging information provided to it.



```
array(1) { ["items"]=> array(3) { [0]=> string(12) "Pack luggage" [1]=> string(13) "Go to airport" [2]=> string(18) "Arrive in San Juan" } }
```

dd() function output

The Laravel Logger

While the `dd()` helper function is useful for quick evaluation of a variable's contents, taking advantage of Laravel's logging facilities is a more effective approach if you plan on repeatedly monitoring one or several data structures or events without necessarily interrupting script execution. Laravel will by default log error-related messages to the application log, located at `storage/logs/laravel.log`. Because Laravel's logging features are managed by Monolog[53], you have a wide array of additional logging options at your disposal, including the ability to write log messages to this log file, set logging levels, send log output to the Firebug console[54] via FirePHP[55], to the Chrome console[56] using Chrome Logger[57], or even trigger alerts via e-mail, HipChat[58] or Slack[59]. Further, if you're using the Laravel 4 Debugbar (introduced later in this chapter) you can easily peruse these messages from the Debugbar's `Messages` tab.

Generating a custom log message is easy, done by embedding one of several available logging methods into the application, passing along the string or variable you'd like to log. Open the `app/Http/Controllers/WelcomeController.php` file and modify the `index` method to look like this:

```
public function index()
{

    $items = [
      'Pack luggage',
      'Go to airport',
      'Arrive in San Juan'
    ];
    \Log::debug($items);
    return view('welcome');

}
```

Save the changes, reload `http://localhost:8000`, and a log message similar to the following will be appended to `storage/logs/laravel.log`:

```
[2015-01-08 01:51:56] local.DEBUG: array (
  0 => 'Pack luggage',
  1 => 'Go to airport',
  2 => 'Arrive in San Juan',
)
```

The debug-level message is just one of several at your disposal. Among other levels are info, warning, error and critical, meaning you can use similarly named methods accordingly:

[53] https://github.com/Seldaek/monolog
[54] https://getfirebug.com/
[55] http://www.firephp.org/
[56] https://developer.chrome.com/devtools/docs/console
[57] http://craig.is/writing/chrome-logger
[58] http://hipchat.com/
[59] https://www.slack.com/

```
\Log::info('Just an informational message.');
\Log::warning('Something may be going wrong.');
\Log::error('Something is definitely going wrong.');
\Log::critical('Danger, Will Robinson! Danger!');
```

Integrating the Logger and FirePHP

When monitoring the log file it's common practice to use the `tail -f` command (available on Linux and OS X) to view any log file changes in real time. You can however avoid the additional step of maintaining an additional terminal window for such purposes by instead sending the log messages to the Firebug[60] console, allowing you to see the log messages alongside your application's browser output. You'll do this by integrating FirePHP[61].

You'll first need to install the Firebug and FirePHP[62] extensions, both of which are available via Mozilla's official add-ons site. After restarting your browser, you can begin sending log messages directly to the Firebug console like so:

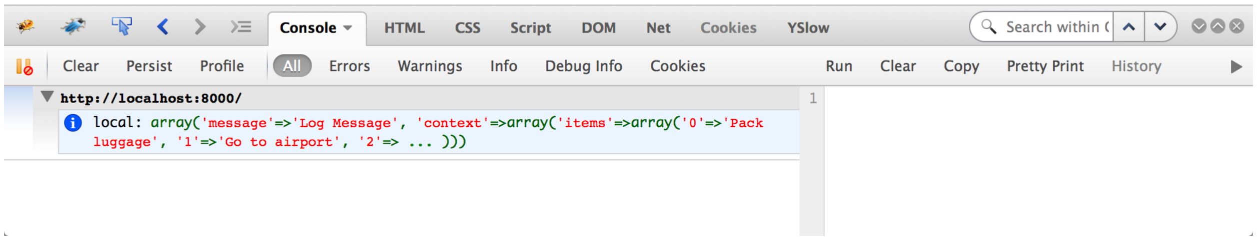
```
$monolog = \Log::getMonolog();

$items = ['Pack luggage', 'Go to airport', 'Arrive in San Juan'];

$monolog->pushHandler(new \Monolog\Handler\FirePHPHandler());

$monolog->addInfo('Log Message', array('items' => $items));
```

Once executed, the `$items` array will appear in your Firebug console as depicted in the below screenshot.



```
http://localhost:8000/
local: array('message'=>'Log Message', 'context'=>array('items'=>array('0'=>'Pack
luggage', '1'=>'Go to airport', '2'=> ... )))
```

Logging to Firebug via FirePHP

Using the Tinker Console

You'll often want to test a small PHP snippet or experiment with manipulating a particular data structure, but creating and executing a PHP script for such purposes is kind of tedious. You can

[60]https://getfirebug.com/
[61]http://www.firephp.org/
[62]https://addons.mozilla.org/en-US/firefox/addon/firephp/

eliminate the additional overhead by instead using the tinker console, a command line-based window into your Laravel application. Open tinker by executing the following command from your application's root directory:

```
$ php artisan tinker --env=local
Psy Shell v0.5.2 (PHP 5.5.21 â€” cli) by Justin Hileman
>>>
```

Notice tinker uses PsySH[63], a great interactive PHP console and debugger. PsySH is new to Laravel 5, and is a huge improvement over the previous console. Be sure to take some time perusing the feature list on the PsySH website[64] to learn more about what this great utility can do. In the meantime, let's get used to the interface:

```
>>> $items = ['Pack luggage', 'Go to airport', 'Arrive in San Juan'];
=> [
       "Pack luggage",
       "Go to airport",
       "Arrive in San Juan"
   ]
```

From here you could for instance learn more about how to sort an array using PHP's `sort()` function:

```
>>> var_dump($items);
array(3) {
  [0]=>
  string(12) "Pack luggage"
  [1]=>
  string(13) "Go to airport"
  [2]=>
  string(18) "Arrive in San Juan"
}
=> null
>>> sort($items);
=> true
>>> $items;
=> [
       "Arrive in San Juan",
       "Go to airport",
       "Pack luggage"
   ]
>>>
```

[63] http://psysh.org/
[64] http://psysh.org/

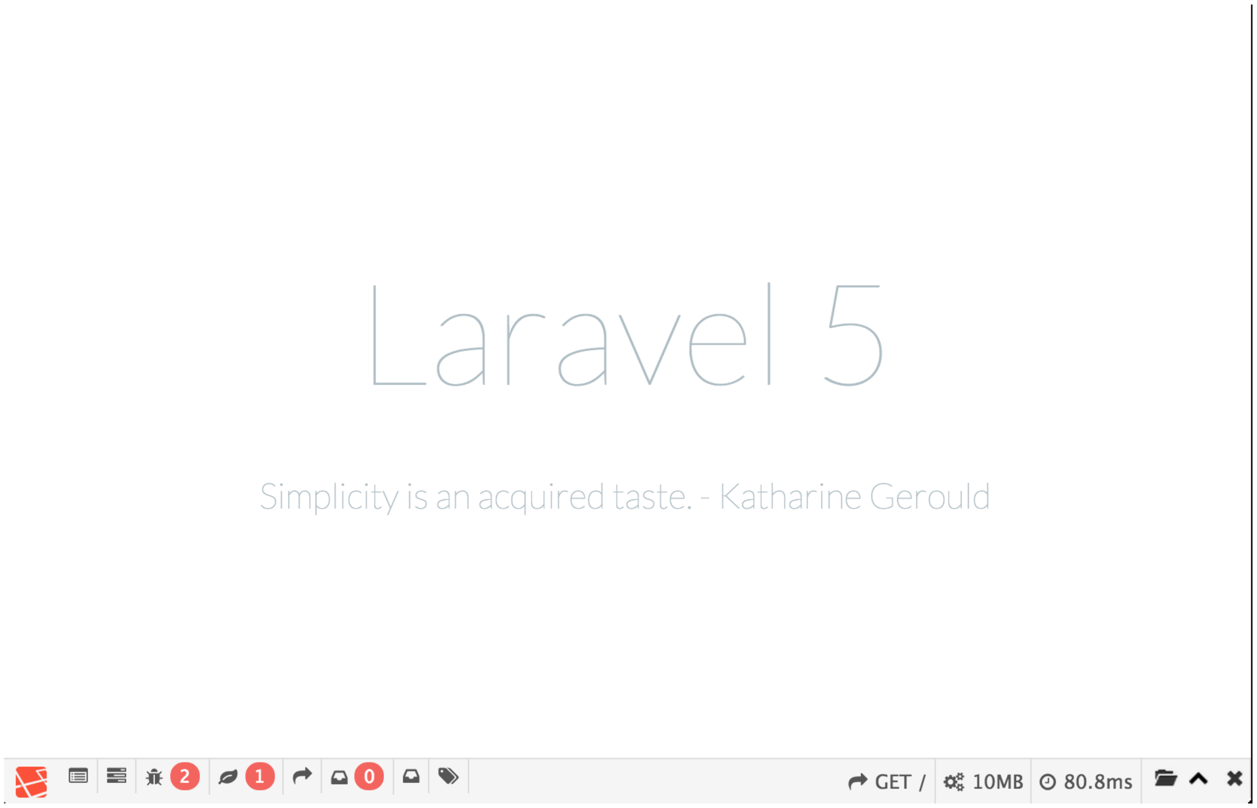
After you're done, type `exit` to exit the PsySH console:

```
>>> exit
Exit:  Goodbye.
$
```

PsySH can be incredibly useful for quickly experimenting with PHP snippets, and I'd imagine you'll find yourself repeatedly returning to this indispensable tool. We'll take advantage of PsySH throughout the book to get acquainted with various Laravel features.

Introducing the Laravel Debugbar

It can quickly become difficult to keep tabs on the many different events that are collectively responsible for assembling the application response. You'll regularly want to monitor the status of database requests, routing definitions, view rendering, e-mail transmission and other activities. Fortunately, there exists a great utility called Laravel Debugbar[65] that provides easy access to the status of these events and much more by straddling the bottom of your browser window (see below screenshot).



The Laravel Debugbar

[65]https://github.com/barryvdh/laravel-debugbar

The Debugbar is visually similar to Firebug[66], consisting of multiple tabs that when clicked result in context-related information in a panel situated below the menu. These tabs include:

- **Messages**: Use this tab to view log messages directed to the Debugbar. I'll show you how to do this in a moment.
- **Timeline**: This tab presents a summary of the time required to load the page.
- **Exceptions**: This tab displays any exceptions thrown while processing the current request.
- **Views**: This tab provides information about the various views used to render the page, including the layout.
- **Route**: This tab presents information about the requested route, including the corresponding controller and action.
- **Queries**: This tab lists the SQL queries executed in the process of serving the request.
- **Mails**: This tab presents information about any e-mails delivered while processing the request.
- **Request**: This tab lists information pertinent to the request, including the status code, request headers, response headers, and session attributes.

To install the Laravel Debugbar, execute the following command:

```
$ composer require barryvdh/laravel-debugbar
Using version ^2.0 for barryvdh/laravel-debugbar
./composer.json has been updated
> php artisan clear-compiled
Loading composer repositories with package information
Updating dependencies (including require-dev)
...
$
```

Next, add the following lines to the `providers` and `aliases` arrays to your `config/app.php` file, respectively:

```
'providers' => [
    ...
    Barryvdh\Debugbar\ServiceProvider::class,
],

...

'aliases' => [
    ...
    'Debugbar' => Barryvdh\Debugbar\Facade::class,
]
```

Save the changes and finally, install the package configuration to your `config` directory:

[66] http://getfirebug.com

```
$ php artisan vendor:publish
```

While you don't have to make any changes to this configuration file (found in `config/debugbar.php`), I suggest having a look at it to see what changes are available.

Reload the browser and you should see the Debugbar at the bottom of the page! Keep in mind the Debugbar will only render when used in conjunction with an endpoint that actually renders a view to the browser.

The Laravel Debugbar is tremendously useful as it provides easily accessible insight into several key aspects of your application. Additionally, you can use the Messages panel as a convenient location for viewing log messages. Logging to the Debugbar is incredibly easy, done using the Debugbar facade. Add the following line to the `Welcome` controller's `index` action (`app/Http/Controllers/WelcomeController.php`):

```
\Debugbar::error('Something is definitely going wrong.');
```

Save the changes and reload the home page within the browser. Check the Debugbar's `Messages` panel and you'll see the logged message! Like the Laravel logger, the Laravel Debugbar supports the log levels defined in PSR-3[67], meaning methods for debug, info, notice, warning, error, critical, alert and emergency are available.

Testing Your Laravel Application with PHPUnit

Automated testing is a critical part of today's web development workflow, and should not be ignored even for the most trivial of projects. Fortunately, the Laravel developers agree with this mindset and automatically include reference the PHPUnit package within every new Laravel project's `composer.json` file:

```
"require-dev": {
  "phpunit/phpunit": "~4.0"
},
```

Laravel 5 includes support for a second testing framework called phpspec[68]. This book doesn't currently include phpspec coverage (pun not intended, I swear!), however stay tuned as a forthcoming release will include an introduction to the topic in the context of Laravel.

Running Your First Test

PHPUnit is a command-line tool that when installed via your project's `composer.json` file is found in `vendor/bin`. Therefore to run PHPUnit you'll execute it like this:

[67]http://www.php-fig.org/psr/psr-3/
[68]http://www.phpspec.net/

```
$ vendor/bin/phpunit --version
PHPUnit 4.7.2 by Sebastian Bergmann and contributors.
```

If you find typing `vendor/bin/` to be annoying, consider making PHPUnit globally available, done using Composer's `global` modifier. Rob Allen has written up a concise tutorial[69] showing you how this is accomplished.

Inside the `tests` directory you'll find a file named `ExampleTest.php` that includes a simple unit test. This test accesses the project home page, and determines whether a 200 status code is returned:

```php
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $this->visit('/')
             ->see('Laravel 5');
    }
}
```

This slick testing API is available as of Laravel 5.1. As you can see, the syntax is very readable and understandable. This example test accesses the project home page and confirms that the text `Laravel 5` is found somewhere on the page. To run the test, just execute the `phpunit` command:

[69]http://akrabat.com/php/global-installation-of-php-tools-with-composer/

```
$ vendor/bin/phpunit
PHPUnit 4.7.2 by Sebastian Bergmann and contributors.

.

Time: 615 ms, Memory: 12.00Mb

OK (1 test, 2 assertions)
```

See that single period residing on the line by itself? That represents a passed test, in this case the test defined by the `testBasicExample` method. If the test failed, you would instead see an `F` for error. To see what a failed test looks like, open up `app/Http/routes.php` and comment out the following lines:

```
$router::get('/', function() {
  return view('welcome');
});
```

I'll introduce the `app/Http/routes.php` file in the next chapter, so don't worry if you don't understand what a route definition is; just understand that by commenting out this line you will prevent Laravel from being able to serve the home page. Save the changes and execute `phpunit` anew:

```
$ vendor/bin/phpunit
PHPUnit 4.7.2 by Sebastian Bergmann and contributors.

F

Time: 403 ms, Memory: 11.75Mb

There was 1 failure:

1) ExampleTest::testBasicExample
A request to [http://localhost] failed. Received status code [404].
```

This time the `F` is displayed, because the assertion defined in `testBasicExample` failed. Additionally, information pertaining to why the test failed is displayed. In the chapters to come we will explore other facets of PHPUnit and write plenty of additional tests.

Consider spending some time exploring the Laravel[70] documentation to learn more about the syntax available to you. In any case, be sure to uncomment that route definition before moving on!

[70]http://laravel.com/docs/master/testing

Conclusion

It’s only the end of the first chapter and we’ve already covered a tremendous amount of ground! With your project generated and development environment configured, it’s time to begin building the TODOParrot application. Onwards!