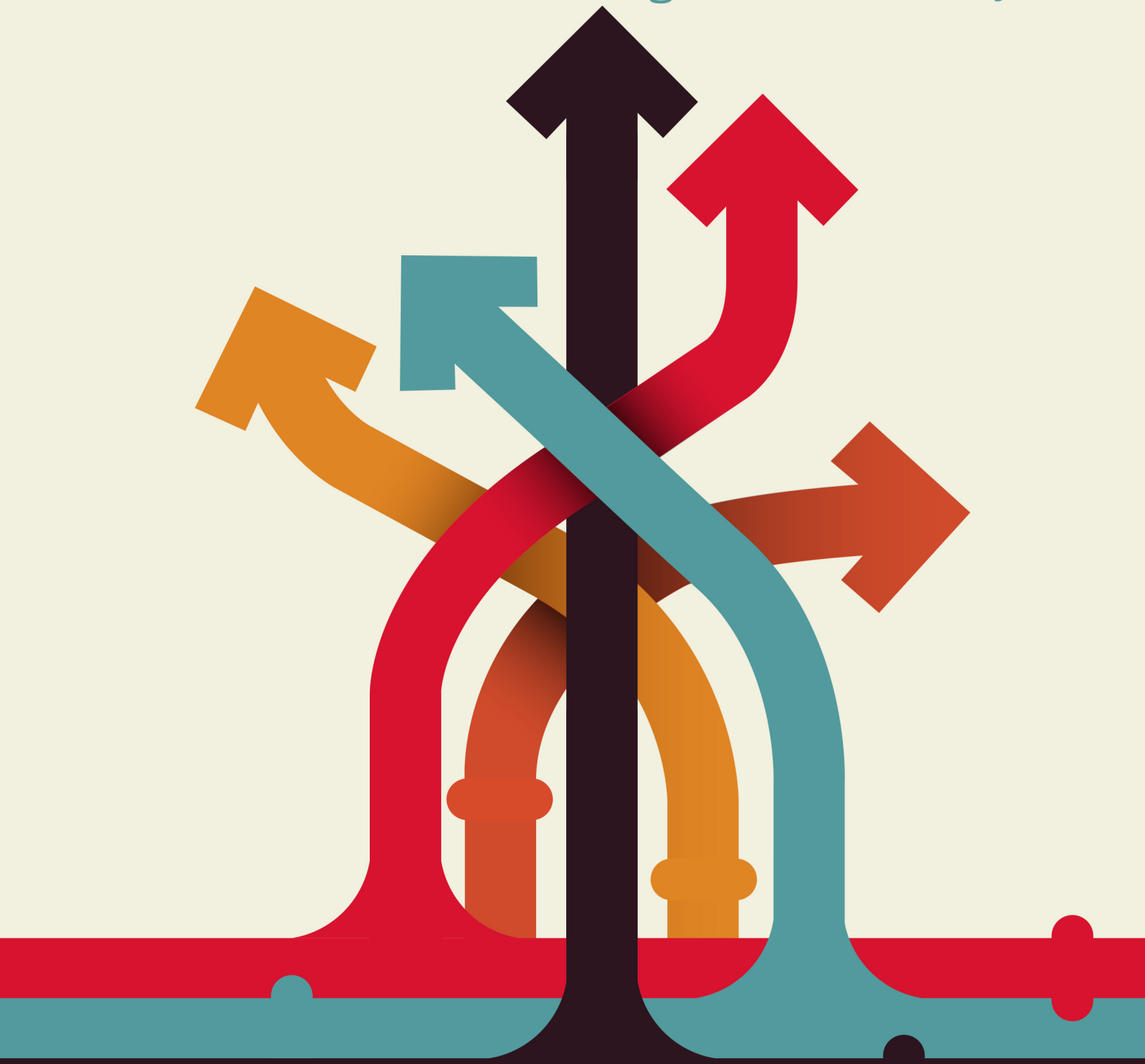
EASY LARAVEL 5

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 2016-10-10

Also By W. Jason Gilmore

Easy Active Record for Rails Developers

Easy E-Commerce Using Laravel and Stripe

Easy React

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

Contents

Introduction

ATTENTION: I'm in the middle of a major book update for Laravel 5.3. The first five chapters have been updated, and I continue working on chapter 6 and beyond. As a reader, you'll always receive free updates so stay tuned for update notifications!

I've spent the vast majority of the past 16 years immersed in the PHP language. During this time I've written eight PHP-related books, including a bestseller that has been in print for more than eleven 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 personally delivered training sessions to hundreds of developers. So you might be surprised to learn a few years ago I became rather disenchanted with the very language that for so very long had consumed the better part of my professional career. 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, old habits are hard to break and so during this time I kept tabs on the PHP community, watching with great interest as several talented developers worked tirelessly to inject that missing enthusiasm back into the language. During my time in the wilderness 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 quickly soared in popularity to become the most followed PHP project on GitHub, quickly surpassing projects and frameworks that had been under active development 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 a relative newcomer to the PHP framework landscape, Laravel is incredibly polished, offering a shallow learning curve, easy test integration, a great object-relational mapping solution called Eloquent, and a wide variety of other great features. 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/

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[5] 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.
- **Elixir**: Elixir[6] offers Laravel users a convenient way to automate various development tasks using Gulp[7], 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[8], 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 form data. In chapter 5 I'll introduce you to this great new feature.
- **Middleware**: 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. You'll learn all about Laravel authentication in chapter 6.
- **Event handling**: Laravel 5 event handlers allow you to reduce redundant logic otherwise found in your controllers by packaging bits of logic separately and then executing that logic in conjunction following certain events, such as sending an e-mail following the registration of a new user. In chapter 11 you'll learn how to create an event handler and then integrate a corresponding event listener into your code.
- **The Lumen Microframework**: Although not part of the Laravel framework per se, Lumen is an optimized version of Laravel useful for creating incredibly fast micro-services and REST APIs. I'll introduce you to this great framework in chapter 10.

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

But we're not going to stop with a mere introduction to these new features. I want you to learn how to build *real-world* Laravel applications, and so I additionally devote extensive coverage to about topics such as effective CSS and JavaScript integration, automated testing, and more!

About this Book

This book is broken into twelve chapters and an appendix, each of which is briefly described below.

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[9] 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.

[9] http://gulpjs.com/

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.

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. 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 9. 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 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's fascinating event broadcasting capabilities, accompanied by a real-world example.

Chapter 12. Introducing Vue.js

Vue.js[10] has become the Laravel community's de facto JavaScript library, and for good reason; it shares many of the practical, productive attributes Laravel developers have come to love. Chapter 12 introduces Vue.js' fundamental features, and shows you how to integrate highly interactive and eye-appealing interfaces into your Laravel application.

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.

[10] http://vuejs.org/

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.

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 SaaS for the interior design and architecture industries, an e-commerce analytics application for a globally recognized publisher, an intranet application for a major South American avocado farm, and a 10,000+ product online store.

Jason is the author of eight books, including the bestselling *Beginning PHP and MySQL, Fourth Edition*, *Easy E-Commerce Using Laravel and Stripe* (with co-author Eric L. Barnes), and *Easy Active Record for Rails Developers*.

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.

[11] http://todoparrot.com
[12] https://github.com/wjgilmore/todoparrot
[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. Newcomers to framework-driven development will have a slightly steeper learning curve due to the introduction of new concepts, however I promise Laravel's practical and user-friendly features will make your journey an enjoyable one.

In this chapter you'll learn how to install Laravel and how to manage your projects using either the Homestead virtual machine or Valet development environment. We'll also create the companion project which will serve as the basis for introducing new concepts throughout the remainder of the book. I'll also introduce you to several powerful debugging and development tools crucial to efficient Laravel development. Finally, you'll learn a bit about Laravel's automated test environment, and how to write automated tests to ensure your application is operating precisely as expected.

Installing Laravel

The easiest way to install Laravel is via PHP's Composer package manager (https://getcomposer.org). If you're not already using Composer to manage your PHP application dependencies, it's easily installed on all major platforms (OS X, Linux, and Windows among them), so head over to the website and take care of that first before continuing.

With Composer installed, run the following command to install Laravel:

```
1  $ composer global require "laravel/installer"
```

After installing the Laravel installer, you'll want to add the directory `~/.composer/vendor/bin` to your system path so you can execute the `laravel` command anywhere within the operating system. The process associated with updating the system path is operating system-specific however a quick Google search will produce all of the instructions you need.

With the system path updated, open a terminal and execute the following command:

```
1  $ laravel --version
2  Laravel Installer version 1.3.1
```

You'll primary use the `laravel` CLI to generate new Laravel projects, which you can do with the `new` command:

```
$ laravel new igniterental
Crafting application...
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
...
Application ready! Build something amazing.
```

If you peek inside the `igniterental` directory you'll see all of the files and directories which comprise a Laravel application! While I know diving into the code found in this newly generated project is very tantalizing, I *implore* you to be patient and take time to first learn more about how to manage your locally hosted Laravel projects. *Homestead* and *Valet* are Laravel's two standard solutions, and I'll introduce you to both next.

Managing Your Laravel Project Development Environment

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 5.6.4 or newer and one of Laravel's supported databases (MySQL, PostgreSQL, SQLite, and Microsoft SQL Server). While those of you who are seasoned PHP developers likely already have local versions of this software installed on your development laptop, I'd like to recommend two *far more efficient* solutions which completely eliminate the need to manage this software on your own. Fortunately for newcomers these solutions will be equally welcome since it allows you to avoid the often time-consuming and error-prone process of installing and configuring PHP, MySQL, and a web server. These solutions are *Homestead* and *Valet*, and in this section you'll learn about both.

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 NGINX[15], a database server such as MySQL[16] or PostgreSQL[17], and often a variety of supplemental technologies such as Redis[18] and Grunt[19]. 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.

[15] http://nginx.org/
[16] http://www.mysql.com/
[17] http://www.postgresql.org/
[18] http://redis.io/
[19] http://gruntjs.com/

In recent years *virtual machines* dramatically lowered this bar. 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 incredible technology, because for instance you can use a virtual machine to run an Ubuntu Linux server on your Windows 10 laptop, 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[20], a virtual machine which bundles everything you need to get started building Laravel-driven websites.

Homestead is currently based on Ubuntu 16.04, and includes everything you need to get started building Laravel applications, including PHP 7.0, Nginx, MySQL, PostgreSQL and a variety of other useful utilities such as Redis and Memcached. It runs flawlessly on OS X, Linux and Windows, and the installation process is very straightforward, meaning in most cases you'll be able to begin managing Laravel applications in less than 30 minutes.

Mac users have another option at their disposal when it comes to locally hosting a Laravel application. It's called Valet, and later in this chapter I'll introduce Valet. If you're looking for a no-frills hosting environment during the development process, Valet is way to go although for most readers I still recommend spending the extra time and effort required to install Homestead because of all the additional features it has to offer.

Installing Homestead

Homestead requires Vagrant[21] and VirtualBox[22]. 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://atlas.hashicorp.com/laravel/homestead
This box can work with multiple providers! The providers that it
can work with are listed below. Please review the list and choose
the provider you will be working with.

1) virtualbox
2) vmware_desktop

```

[20] http://laravel.com/docs/homestead
[21] http://www.vagrantup.com/
[22] https://www.virtualbox.org/wiki/Downloads

```
Enter your choice: 1
==> box: Adding box 'laravel/homestead' (v0.4.2) for provider: virtualbox
    box: Downloading: https://atlas.hashicorp.com/laravel/boxes/homestead/versio\
ns/0.4.2/providers/virtualbox.box
==> box: Successfully added box 'laravel/homestead' (v0.4.2) for 'virtualbox'!
```

Throughout the book I'll use the $ symbol to represent 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 hundreds of different boxes useful for building applications using a wide variety of technology stacks, so check out this list of popular boxes[23] for an idea of what else is available.

Once the box has been added, you'll next want to install Homestead. To do so, you'll ideally using Git to clone the repository. If you don't already have Git installed you can easily do so by heading over to the Git website[24] or using your operating system's package manager.

Next, open a terminal and enter your home directory:

```
$ cd ~
```

Then use Git's `clone` command to clone the Homestead repository:

```
$ git clone https://github.com/laravel/homestead.git Homestead
Cloning into 'Homestead'...
remote: Counting objects: 1497, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 1497 (delta 0), reused 0 (delta 0), pack-reused 1492
Receiving objects: 100% (1497/1497), 241.74 KiB | 95.00 KiB/s, done.
Resolving deltas: 100% (879/879), done.
Checking connectivity... done.
```

You'll see this has resulted in the creation of a directory named `Homestead` in your home directory which contains the repository files. Next, you'll want to enter this directory and execute the following command:

[23] https://vagrantcloud.com/discover/popular
[24] https://git-scm.com/downloads

```
$ bash init.sh
```

If you're on Windows you'll instead want to run the following command:

```
$ init.bat
```

This will create a directory called `.homestead`, which will also be placed in your home directory. You'll modify the files found in this directory to configure Homestead in a variety of ways, including most notably how to find and serve the web applications which will be hosted on the virtual machine.

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[25] offers a succinct set of steps. If you're running Linux or OS X, nixCraft[26] 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 which looks like this:

```
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 the `keys` property. 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:

[25] http://kb.siteground.com/how_to_generate_an_ssh_key_on_windows_using_putty/
[26] 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
```

This particular step tends to be the source of great confusion Homestead beginners, so pay close attention to the following description. The `folders` structure'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* identify your Laravel project's root directory (I realize we haven't created the project or directory just yet, so just keep in mind this value must identify that soon-to-be-created directory). The `folders` structure'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. You almost certainly do not have to change the `to` attribute's default value, so don't worry about it for now.

Windows users should keep in mind the tilde (~) home directory shortcut is not supported on Windows and so you'll need to specify the absolute path to your chosen directory.

The `sites` structure's `map` attribute defines the domain name you'll use to access the Laravel application via the browser. For instance, you might change this to `dev.todoparrot.com`. Keep in mind this domain name is used purely for internal developmental purposes, so you don't actually have to own the domain name.

Finally, the `sites` structure's `to` attribute defines the Laravel project's root *web directory*, which is `/public` by default. This isn't just some contrived setting; a file named `index.php` resides in your Laravel application's `/public` directory, and it "listens" for incoming requests to your application and kicks off the process which ultimately results in the client's desired resource (web page, JSON data, etc.) being returned.

Despite my best efforts this explanation is likely clear as mud, so let's clarify with an example. Begin by setting the `folders` structure's `map` attribute to somewhere within the directory where you tend to manage your various software projects. For instance, mine is set like this:

```
folders:
    - map: ~/Code/dev.todoparrot.com
    - to: /home/vagrant/Code
```

Next, modify the `sites` structure to look like this:

```
sites:
    - map: dev.todoparrot.com
      to: /home/vagrant/Code/dev.todoparrot.com/public
```

Save the changes and we'll next create a quick test to confirm you can indeed talk to the Homestead webserver. Create 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 from within your `Homestead` directory:

```
$ vagrant 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: Setting the name of the VM: homestead-7
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
...
==> 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)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
$
```

Your Homestead virtual machine is up and running! With that done, we have one remaining step. We'll need to configure your laptop to recognize what it should do when the `dev.todoparrot` URL defined in `Homestead.yaml` is requested in the browser. To do so, you'll need 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  dev.todoparrot.com
```

After saving these changes, we'll want to create the Laravel project that will be served via this URL. However, there still remains plenty to talk about regarding Homestead and virtual machine management so in the sections that follow I discuss several important matters pertaining to this topic. For the moment though I suggest jumping ahead to the section "Creating the TODOParrot Application" and returning to the below sections later.

Managing Your Virtual Machine

There are a few administrative tasks you'll occasionally need to carry out regarding management of your virtual machine. For example, if you'd like to shut down the virtual machine you can do so using the following command:

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

To later boot the machine back up, you can execute `vagrant up` as we did previously:

```
$ vagrant up
```

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

```
$ vagrant destroy
```

I stress executing the `destroy` command this *will delete* the virtual machine and all of its data! Executing this command is very different from shutting down the machine using `halt`.

If you happen to have installed more than one box (it can be addictive), use the `box list` command to display them:

```
$ vagrant box list
laravel/homestead (virtualbox, 0.4.2)
```

These are just a few of the many commands available to you. Run `vagrant --help` for a complete listing of what's available:

```
$ vagrant --help
```

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. If you're running Linux or OS X, you can SSH into the virtual machine using the `ssh` command:

```
$ ssh vagrant@127.0.0.1 -p 2222
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Thu Mar 10 17:11:55 2016 from 10.0.2.2
```

Windows users will need to first install an SSH client. A popular Windows SSH client is PuTTY[27].

In either case, 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:

```
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.

Transferring Files Between Homestead and Your Laptop

If you create a file on a Homestead and would like to transfer it to your laptop, you have two options. The easiest involves SSH'ing into Homestead and moving the file into one of your shared directories, because the file will instantly be made available for retrieval via your laptop's file system. For instance if you're following along with the `dev.todoparrot.com` directory configuration, you can SSH into Homestead, move the file into `/home/vagrant/dev.todoparrot.com`, and then logout of SSH. Then using your local terminal, navigate to `~/Code/dev.todoparrot.com` and you'll find the desired file sitting in your local `dev.todoparrot.com` root directory.

Alternatively, you can use `sftp` to login to Homestead, navigate to the desired directory, and transfer the file directly:

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

```
$ sftp -P 2222 vagrant@127.0.0.1
Connected to 127.0.0.1.
sftp> cd dev.farm.com
sftp> get hello.txt
Fetching /home/vagrant/dev.todoparrot.com/db.sql.gz to db.sql.gz
/home/vagrant/dev.farm.com/db.sql.gz 0%    0     0.0KB/s   --:-- ETA
sftp>
```

Connecting to Your Database

Although this topic won't really be relevant until we discuss databases in chapter 3, this nonetheless seems a logical place to show you how to connect to your project's Homestead database. If you return to `Homestead.yaml`, you'll find the following section:

```
databases:
    - homestead
```

This section is used to define any databases you'd like to be automatically created when the virtual machine is first booted (or re-provisioned; more about this in the next section). As you can see, a default database named `homestead` has already been defined. You can sign into this database now by SSH'ing into the machine and using the `mysql` client:

```
$ ssh vagrant@127.0.0.1 -p 2222
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed Mar 23 00:56:23 2016 from 10.0.2.2
```

After signing in, enter the database using the `mysql` client, supplying the default username of `homestead` and the desired database (also `homestead`). When prompted for the password, enter `secret`:

```
vagrant@homestead:~$ mysql -u homestead homestead -p
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2330
Server version: 5.7.11 MySQL Community Server (GPL)

```

```
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

There are of course no tables in the database (we'll do this in chapter 3), but feel free to have a look anyway:

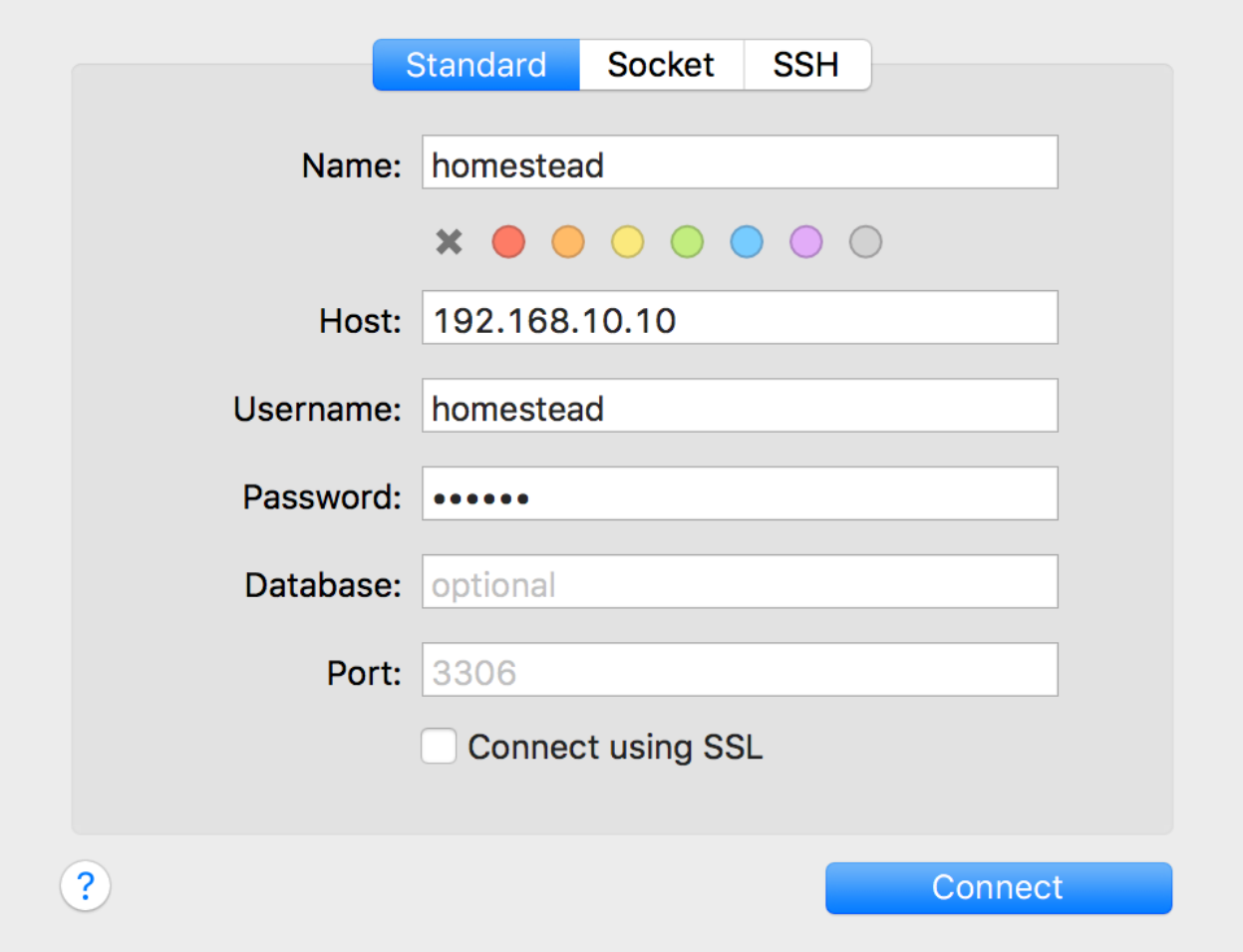
```
mysql> show tables;
Empty set (0.00 sec)
```

To exit the `mysql` client, just execute `exit`:

```
mysql> exit;
Bye
$
```

Chances are you prefer to interact with your database using a GUI-based application such as Sequel Pro[28] or phpMyAdmin[29]. You'll connect to the `homestead` database like you would any other, by supplying the username (`homestead`), password (`secret`), and the host, which is `192.168.10.10`. For instance, the following screenshot depicts my Sequel Pro connection window:

[28] http://www.sequelpro.com/
[29] https://www.phpmyadmin.net/



The Sequel Pro connection window

Of course, you may want to change the name of this default database, or define additional databases as the number of projects you manage via Homestead grows in size. I'll show you how to do this next.

Defining Multiple Homestead Sites and Databases

My guess is you'll quickly become so enamored with Homestead that it will be the default solution for managing all of your Laravel projects. This means you'll need to define multiple projects within the `Homestead.yaml` file. Fortunately, doing so is easier than you think. Check out the following slimmed down version of my own `Homestead.yaml` file, which defines two projects (`dev.larabrain.com` and `dev.todoparrot.com`):

```
folders:
    - map: ~/Code/dev.larabrain.com
      to: /home/vagrant/dev.larabrain.com
    - map: ~/Code/dev.todoparrot.com
      to: /home/vagrant/dev.todoparrot.com

sites:
    - map: dev.larabrain.com
      to: /home/vagrant/dev.larabrain.com/public
    - map: dev.todoparrot.com
      to: /home/vagrant/dev.todoparrot.com/public

databases:
    - dev_larabrain_com
    - dev_todoparrot_com
```

Notice how I've also defined two different databases, since each application will logically want its own location to store data.

After saving these changes, you'll want your virtual server to be reconfigured accordingly. If you have never started your virtual server, running `vagrant up` will be suffice because the `Homestead.yaml` file had never previously been read. However, if you've already started the VM then you'll need to force Homestead to *reprovision* the virtual machine. This involves reloading the configuration. To do so, you'll first need to find the identifier used to present the currently running machine:

```
$ vagrant global-status
id       name    provider   state   directory
------------------------------------------------------------------
6f13a59  default virtualbox running /Users/wjgilmore/Homestead
```

Copy and paste that `id` value (`6f13a59` in my case), supplying it as an argument to the following command:

```
$ vagrant reload --provision 6f13a59
==> default: Attempting graceful shutdown of VM...
==> default: Checking if box 'laravel/homestead' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
...
```

Once this command completes, your latest `Homestead.yaml` changes will be in place!

Introducing Valet

Virtual machines and ready-made environments such as Homestead are great, and have become indispensable tools I use on a daily basis. However Homestead can admittedly be rather intimidating for newcomers, and can frankly be overkill for many developers. If you use a Mac and are interested in a no-frills development environment, Laravel offers a streamlined solution called *Valet* which can be configured in mere minutes.

Installing Valet

To install Valet you'll need to first install Homebrew (http://brew.sh/), the community-driven package manager for OS X. As you'll see on the home page, Homebrew is very easy to install and should only take a moment to complete. Once done, you'll want to install PHP 7. You can do so by executing the following command:

```
1 $ brew install homebrew/php/php70
```

Next you'll install Valet using Composer. Like Homebrew, Composer (https://getcomposer.org) is a package manager but is specific to PHP development, and is similarly easy to install. With Composer installed, install Valet using the following command:

```
1 $ composer global require laravel/valet
```

Finally, configure Valet by running the following command, which among other things will ensure it always starts automatically when you reboot your machine:

```
1 $ valet install
```

Presuming your Laravel applications will use a database, you'll also need to install a database such as MySQL or MariaDB. You can easily install either using Homebrew. For instance, you can install MySQL like so:

```
1 $ brew install mysql
```

After installation completes just follow the instructions displayed in the terminal to ensure MySQL starts automatically upon system boot.

Serving Sites with Valet

With Valet installed and configured, you'll next want to create a directory to host your various Laravel projects. I suggest creating this directory in your home directory; consider calling it something easily recognizable such as `Code` or `Projects`. Enter this directory using your terminal and execute the following command:

```
$ valet park
This directory has been added to Valet's paths.
```

The `park` command tells Valet monitor this directory for Laravel projects, and automatically make a convenient URL available for viewing the project in your browser. For instance, while inside the project directory create a new Laravel project named `todoparrot`:

```
$ laravel new todoparrot
```

After creating the project, open your browser and navigate to `http://todoparrot.dev` and you'll see the project's default splash screen (presented in the following screenshot).

The Laravel splash page

It doesn't get any easier than that!

Creating the TODOParrot Application

With Laravel (and presumably Homestead or Valet) 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. Create a new Laravel project using the `laravel` command:

```
$ laravel new todoparrot
```

Of course, you can call the project directory anything you want. If you're using Valet, then you're free to place the project directory anywhere you please. However, if you're using Homestead, recall Homestead expects the application to reside in the directory you specified within the `Homestead.yaml` file's `folders` structure's `map` property. As a reminder here is what mine looks like:

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

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[30] library to conveniently manage your application's configuration variables. 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, which should be used as a template from which fellow developers will copy over to `.env` and modify to suit their own needs. I'll talk more about these files and practical approaches for managing your environment settings in the later section, "Configuring Your Laravel Application".
- `.gitattributes`: This file is used by Git[31] to ensure consistent settings across machines, which is useful when multiple developers using a variety of operating systems are working on the same project. You'll find a few default settings in the file, however these are pretty standard and you in all likelihood won't have to modify them. Plenty of other attributes are however available; Scott Chacon's online book, "Pro Git"[32] includes a section ("Customizing Git - Git Attributes"[33]) 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 a few default settings in here, including the `vendor` directory which houses the Laravel source code and other third-party packages, and the `.env` file, which should never be managed in version control since it presumably contains sensitive settings such as database passwords.
- `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 book progresses.
- `artisan`: `artisan` is a command-line tool we'll use to rapidly create new parts of your applications such as controllers and models, manage your database's evolution through a great feature known as *migrations*, and interactively debug your application. We'll return to `artisan` repeatedly throughout the book because it is such an integral part of Laravel development.
- `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 this directory.

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

- `composer.json`: Composer[34] is PHP's de facto package manager, used by thousands of developers around the globe to quickly integrate popular third-party solutions such as Swift Mailer[35] and Doctrine[36] into a PHP application. Laravel heavily depends upon 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 by the time you're done reading this book you'll wonder how you ever lived without it. In fact in this introductory chapter alone we'll use it several times to install various useful packages.
- `composer.lock`: This file contains information about the state of your project's 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 several files used to configure various aspects of your Laravel application, such as the database credentials, 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 5.0 introduced a new feature called *Laravel Elixir*. Elixir relies upon the `Gulpfile.js` file to define various Gulp.js[37] tasks useful for automating 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.
- `phpunit.xml`: Even trivial web applications should be accompanied by an automated test suite. Laravel leaves little room for excuse to avoid this best practice by automatically configuring your application to use the popular PHPUnit[38] 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 the topic of testing repeatedly throughout the book.
- `public`: The `public` directory serves as your application's web root directory, housing the `.htaccess`, `robots.txt`, and `favicon.ico` files, in addition to a file named `index.php` that is the *first* 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. It's because the `index.php` file serves as the front controller that you needed to identify the `public` directory as your application's root directory when configuring `Homestead.yaml` earlier in this chapter.
- `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[39] 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 such as CoffeeScript and Saas files.

[34] https://getcomposer.org
[35] http://swiftmailer.org/
[36] http://www.doctrine-project.org/
[37] http://gulpjs.com/
[38] http://phpunit.de/
[39] http://github.com/wjgilmore/todoparrot

- `routes`: The `routes` directory is new to 5.3. It replaces the old `app/Http/routes.php` file, and separates your application's routing definitions into three separate files: `api.php`, `console.php`, and `web.php`. Collectively, these files determine how your application responds to different endpoints. We'll return to these files repeatedly throughout the book, beginning in Chapter 2.
- `server.php`: The `server.php` file can be used to bootstrap your application for the purposes of serving it via PHP's built-in web server. While a nice feature, Homestead offers a far superior development experience and so you can safely ignore this file and feature.
- `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, and thanks to Laravel's incredibly simple test integration features I highly encourage you to follow along closely with the examples provided in these sections.
- `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 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. Presuming you've configured Homestead and generated the Laravel project in the appropriate directory, you should be able to navigate to `http://dev.todoparrot.com` (`http://todoparrot.dev` if you're using Valet) and see the page presented in the below screenshot:

The Laravel splash page

So where does this splash page come from? 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[40], meaning your project controllers, models, and other key resources are namespaced. The default namespace is set to `App`, which is pretty generic, however if you don't plan on distributing your application to third-parties then the default is going to be just fine. If you did want to update your project's namespace to something 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.

Because I'm not distributing IgniteRental as software which might be integrated into an application, I've left the default namespace as `App`.

Configuring Your Laravel Application

Laravel offers 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, and locale.
- `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 the event broadcasting feature, which is useful when you want to simultaneously notify multiple application users of some event such as the addition of a new blog post. I discuss event broadcasting in Chapter 11.

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

- `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. You'll learn all about Laravel's database support in chapters 3 and 4.
- `filesystems.php`: The `filesystems.php` configuration file defines the file system your project will use to manage assets such as file uploads. Thanks to Laravel's integration with Flysystem[41], support is available for a wide variety of adapters, among them the local disk, Amazon S3, Azure, Dropbox, FTP, Rackspace, and Redis.
- `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 (a variety of which are supported, among them Sendmail, SMTP, PHP's `mail()` function, and Mailgun). 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[42] or Amazon Simple Queue Service[43]. 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 Mailgun 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, 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.

[41] https://github.com/thephpleague/flysystem
[42] http://kr.github.io/beanstalkd/
[43] http://aws.amazon.com/sqs/

Configuring Your Environment

Your application will likely require access to database credentials and other sensitive information such as API keys for accessing third party services. This confidential information should never be shared with others, and therefore you'll want to take care it isn't embedded directly into the code. Instead, you'll want to manage this data within *environment variables*, and then refer to these variables within the application.

Laravel supports a very convenient solution for managing and retrieving these variables thanks to integration with the popular PHP dotenv[44] package. When developing your application you'll define environment variables within the `.env` file found in your project's root directory. The default `.env` file looks like this:

```
APP_ENV=local
APP_KEY=base64:Xq9+pNBKPc1IskLbT7M3Y08kzQ=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

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

```

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

```
PUSHER_APP_ID=
PUSHER_KEY=
PUSHER_SECRET=
```

These variables can be retrieved anywhere within your application using the `env()` function. For instance, the `config/database.php` is used to define your project's database connection settings (we'll talk more about this file in Chapter 3). It retrieves the `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD` variables defined within `.env`:

```
'mysql' => [
    'driver' => 'mysql',
    'host' => env('DB_HOST', 'localhost'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'charset' => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix' => '',
    'strict' => true,
    'engine' => null,
],
```

You'll see the `.gitignore` includes `.env` by default. This is because you should *never* manage `.env` in your version control repository! Instead, when it comes time to deploy your application to production, you'll define the variables found in `.env` as *server environment variables* which can also be retrieved using PHP's `env()` function. I'll talk more about managing these variables in your other environments in Chapter 9.

We'll return to the configuration file throughout the book as new concepts and features are introduced.

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 a few of my favorite 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[45], FirePHP[46], and the many tools integrated into PHP IDEs such as Zend Studio[47] and PHPStorm[48].

The dd() Function

Ensuring the `.env` file's `APP_DEBUG` variable is set to `true` is the easiest way to view information about any application errors, because Laravel will dump error- and exception-related information directly to the browser. However, 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()`[49] helper function, which will dump a variable's contents to the browser and halt further script execution. For example suppose you defined an array inside a Laravel application and wanted to output its contents to the browser. Here's an example array:

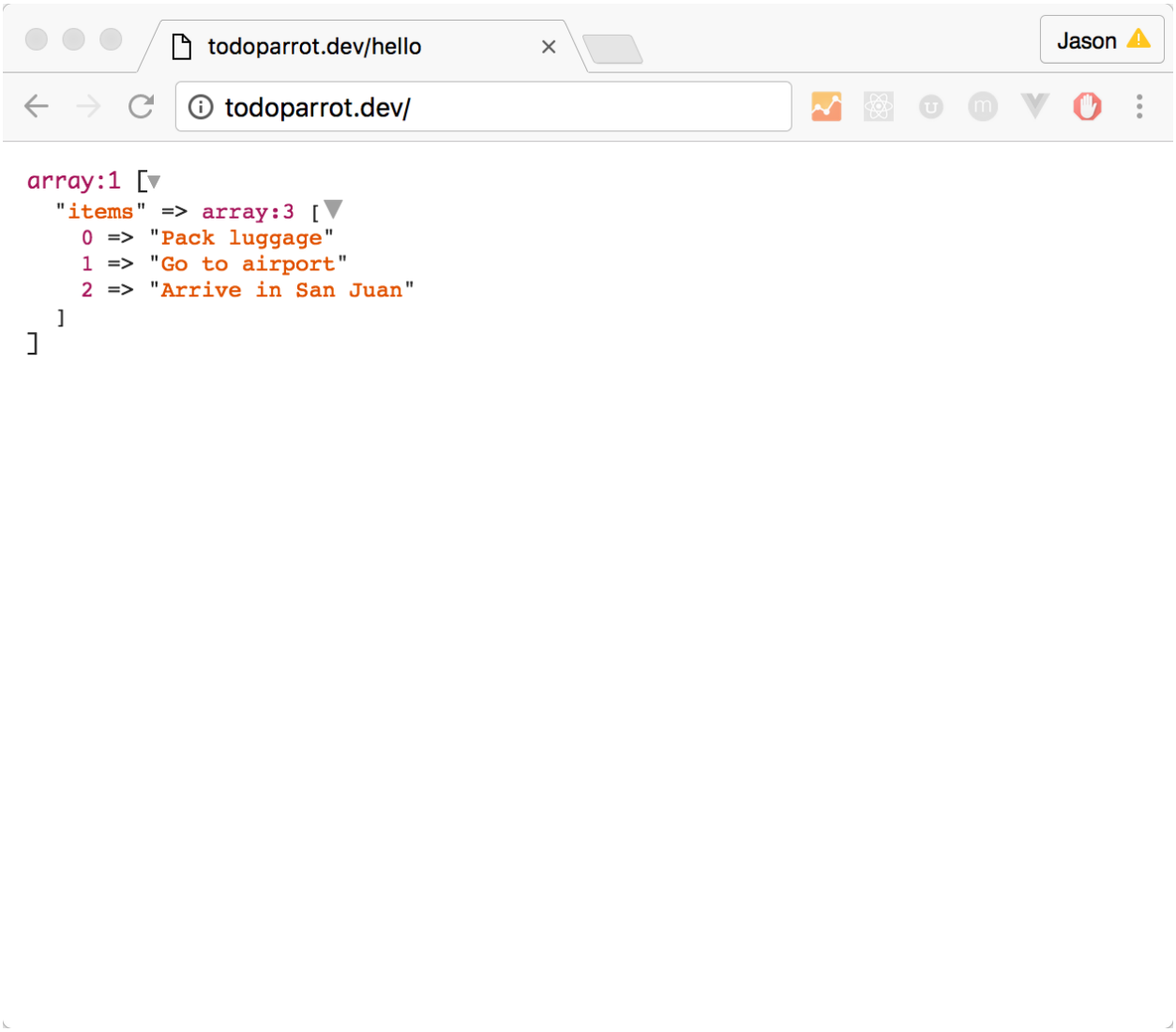
```
$items = [
    'items' => [
      'Pack luggage',
      'Go to airport',
      'Arrive in San Juan'
    ]
];
```

You could execute the `dd()` function like so:

```
dd($items);
```

Passing `$items` into `dd()` will cause the array contents to be dumped to the browser window as depicted in the below screenshot.

[45] http://xdebug.org/
[46] http://www.firephp.org/
[47] http://www.zend.com/en/products/studio
[48] http://www.jetbrains.com/phpstorm/
[49] https://laravel.com/docs/master/helpers#method-dd



dd() function output

Of course, it is likely at this point you don't know where this code would even be executed. Not to worry! In the chapters to come just keep this and the following solutions in mind so you can easily debug your code once we start building the application.

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 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[50], 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[51] via FirePHP[52], to the Chrome console[53] using Chrome Logger[54], or even trigger alerts via e-mail, HipChat[55] or Slack[56]. Further, if you're using the Laravel 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. Returning to the `$items` array, suppose you instead wanted to log its contents to Laravel's log:

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

\Log::debug($items);
```

After reloading the browser to execute this code, you'll see a log message similar to the following will be appended to `storage/logs/laravel.log`:

```
[2016-09-21 09:14:29] local.DEBUG: array (
  'items' =>
  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:

[50] https://github.com/Seldaek/monolog
[51] https://getfirebug.com/
[52] http://www.firephp.org/
[53] https://developer.chrome.com/devtools/docs/console
[54] http://craig.is/writing/chrome-logger
[55] http://hipchat.com/
[56] 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[57] console, allowing you to see the log messages alongside your application's browser output. You'll do this by integrating FirePHP[58].

You'll first need to install the Firebug and FirePHP[59] extensions. If you're running Firefox, both are available via Mozilla's official add-ons site. If you're running another browser such as Chrome, you can install Firebug Lite[60] and FirePHP4Chrome. After restarting your browser, you can begin sending log messages directly to the browser console by adding the following to `bootstrap/app.php`:

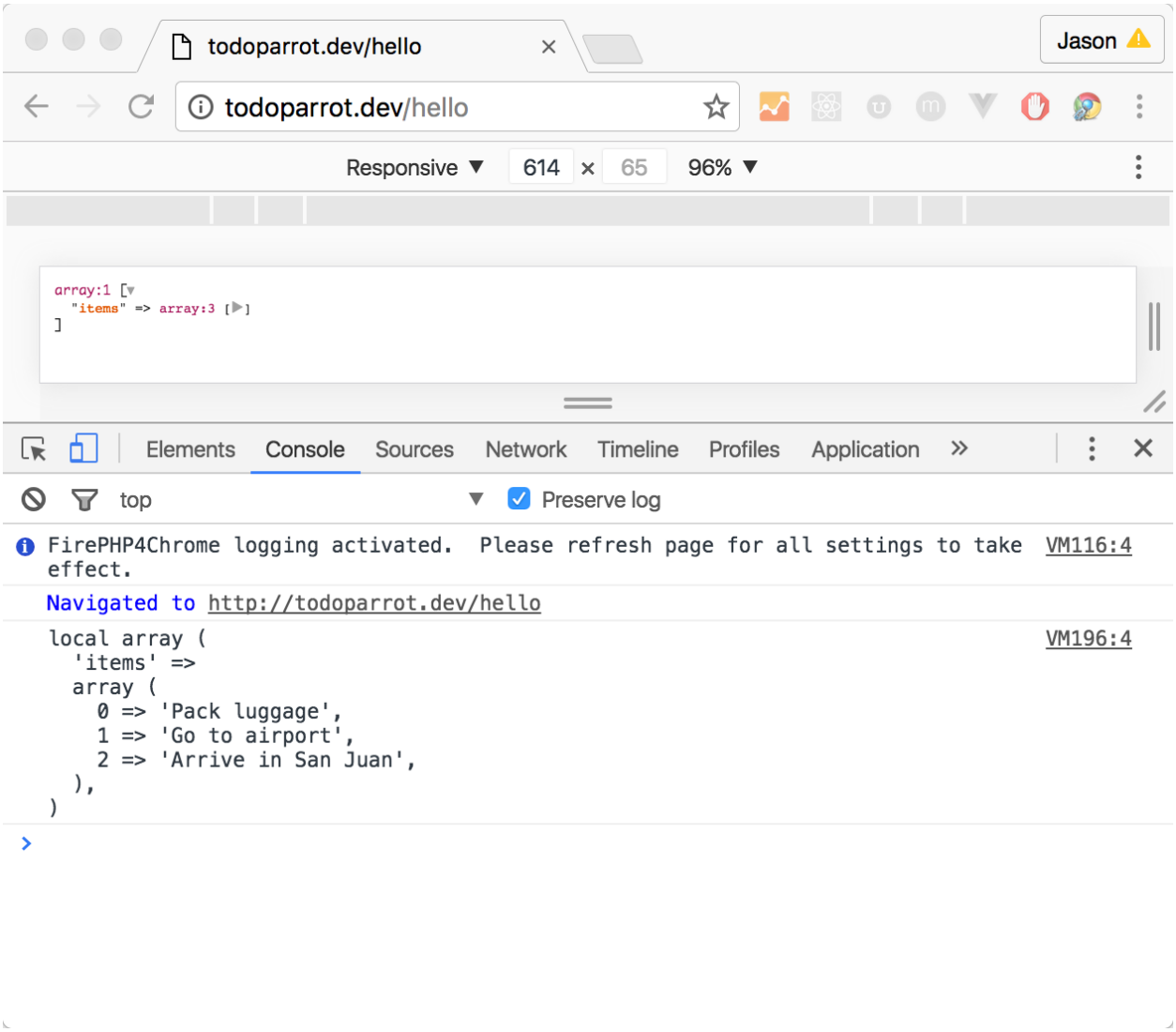
```
$app->configureMonologUsing(function($monolog) {
    $monolog->pushHandler(new \Monolog\Handler\FirePHPHandler());
});
```

After saving the changes, you can log for instance the `$items` array just as you did previously:

```
\Log::debug($items);
```

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

[57] https://getfirebug.com/
[58] http://www.firephp.org/
[59] https://addons.mozilla.org/en-US/firefox/addon/firephp/
[60] https://getfirebug.com/firebuglite



Logging to the Chrome console via Firebug Lite and 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 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
Psy Shell v0.7.2 (PHP 7.0.6 â€” cli) by Justin Hileman
>>>
```

Tinker uses PsySH[61], 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[62] 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:

```
>>> sort($items);
=> true
>>> $items;
=> [
       "Arrive in San Juan",
       "Go to airport",
       "Pack luggage"
   ]
>>>
```

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

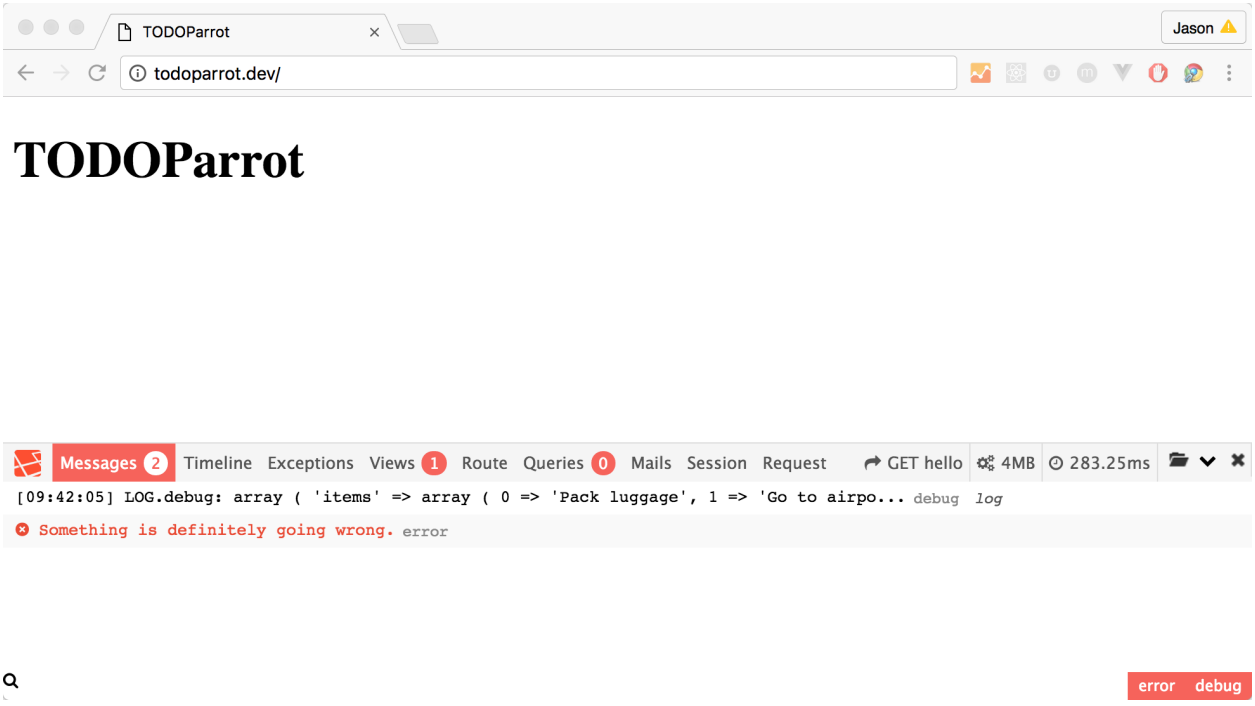
```
>>> exit
Exit:  Goodbye.
$
```

The Tinker console 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 Tinker throughout the book to get acquainted with various Laravel features.

[61] http://psysh.org/
[62] http://psysh.org/

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[63] 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

The Debugbar is visually similar to Firebug[64], 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.

[63] https://github.com/barryvdh/laravel-debugbar
[64] http://getfirebug.com

- **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.
- **Session**: This table presents any session-related information made available 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 --dev
Using version ^2.2 for barryvdh/laravel-debugbar
./composer.json has been updated
Loading composer repositories with package information
...
$
```

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 install the package configuration to your `config` directory:

```
$ 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:

```
\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[65], 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 PHPUnit support with every new Laravel project. PHPUnit is a very popular *unit testing framework* which allows you to create and execute well-organized tests used to confirm all parts of your application are working as expected.

Each new Laravel application even includes an example test which you can use as a reference for beginning to write your own tests! You'll find this test inside the `tests` directory. It's named `ExampleTest.php`, and it demonstrates how to write a test that accesses the project home page, and determines whether the text `Laravel 5` is visible:

```
<?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');
    }
}
```

To run the test, execute the `phpunit` command from within your project's root directory:

[65] http://www.php-fig.org/psr/psr-3/

```
$ phpunit
PHPUnit 5.5.4 by Sebastian Bergmann and contributors.

.                                                   1 / 1 (100%)

Time: 262 ms, Memory: 10.00MB

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 `tests/ExampleTest.php` and locate the following line:

```
->see('PHP');
```

Replace the string `Laravel` with anything you please, such as `PHP`. If you reload the browser after saving the changes you'll see the updated text. Now run execute `phpunit` anew:

```
$ phpunit
PHPUnit 5.5.4 by Sebastian Bergmann and contributors.

F                                                   1 / 1 (100%)

Time: 262 ms, Memory: 10.00MB

There was 1 failure:

1) ExampleTest::testBasicExample
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
...
Failed asserting that the page contains the HTML [PHP].
Please check the content above.

...

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

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[66] documentation to learn more about the syntax available to you. In any case, be sure to uncomment that route definition before moving on!

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 application. Onwards!

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