Laravel has established itself as one of the most popular PHP frameworks over the past couple of years. Its popularity comes due to its ease of use, expressive syntax, and the number of components it has, allowing you to achieve practically any task in a modern PHP application.

If you’ve not had the opportunity to dive into Laravel yet, then this book will be the perfect companion, as it covers the fundamentals Laravel is built on in a thorough yet easy-to-follow manner. To make this book your best learning source, we have covered integral topics such as preparing an environment for working with Laravel applications, creating your first Laravel application from start to finish, a tour of Laravel’s ORM Eloquent, and more advanced topics such as testing, user authentication, and security. Being one of the most interesting features in Laravel, we also cover an overview of Artisan, including descriptions of some of its tasks. Post this, we talk about testing and security. By the end of this book, you will be able to create robust PHP websites and web applications quickly and efficiently.

Who this book is written for
This book is intended for PHP web developers who have an interest in Laravel and who know the basics of the framework in theory, but don’t really know how to use it in practice. No experience of using frameworks is required, but it is assumed you are at least familiar with building dynamic websites in PHP already.

What you will learn from this book
- Set up a development environment for working with Laravel apps
- Create a working Laravel application from start to finish
- Interact with data and discover Eloquent, Laravel’s own ORM
- Create and run migrations and seeds
- Simplify command-line tasks by building Artisan commands
- Protect your app by securing it from common security vulnerabilities
- Perform end-to-end testing after learning the basics of testing in Laravel
- Learn about Laravel’s other features including service providers, packages, queue workers, and helpers for working with strings and arrays
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'An Introduction to Laravel'
- A synopsis of the book’s content
- More information on Laravel 5 Essentials

About the Author

Martin Bean is a full-stack website developer based in Newcastle upon Tyne, UK. Prior to writing this book, he spent 8 years as a professional website developer, beginning his career and honing his skills in various agencies. After 5 years, he made the switch to running his own development and consultancy firm, where he worked with clients, both big and small.

Although this is the first book he has authored, he has written other articles and acted as a technical reviewer on a Node.js title. He also blogs regularly on his own website, http://martinbean.co.uk.

You can follow Martin on Twitter at https://twitter.com/martinbean.
Laravel 5 Essentials

Application frameworks have grown in popularity over the past five years. There has been a tremendous shift from handwriting all code to leveraging these powerful frameworks with prebuilt components and features. However, with anything that comes to be in fashion, there are now a lot of contending options, and each of them viable.

While CodeIgniter was one of the first frameworks to enjoy widespread popularity, this popularity would come to be its undoing years later, as its large spread use and low barrier to entry meant it couldn't take advantage of newer versions of PHP without losing backwards compatibility, and potentially breaking lots of applications. This saw it then be surpassed by faster-moving alternatives such as Symfony and even FuelPHP, which was developed as a response to CodeIgniter's unwillingness to embrace change.

Enter: Laravel. Laravel joined the framework scene when there were already many players. However, the developers of Laravel used this timing to their advantage, instead creating a framework that avoided all of the problems and mistakes previous full stack frameworks had made and building on top of the excellent Symfony components in order to create a robust, component-based framework.

Instead of providing dozens of inflexible libraries, Laravel provides sensible, driver-based components that developers could use to build applications their own way, rather than trying to mash everything into the layout the framework author defined. This led to Laravel rising in popularity. It was also a fast-moving framework, and, by version 4, had become the most starred framework on GitHub, a testament to its popularity.

This book will give you a tour of Laravel and its core features. We'll look at how to manage multiple Laravel applications on the same machine and then we'll go ahead and start building our own Laravel application from scratch through to completion. Once we've got a basic application reading and writing data from a database, we'll take a look at Eloquent, Laravel's ORM, which is what makes it easy to read and write from a database and the more advanced features it offers. From there, we'll look at Artisan, Laravel's command-line utility, and even how to define our own commands. We'll then learn how to write automated tests for our application to make sure it keeps working the way we want it to, even with future developments. Then, finally, we'll look at how to build login and registration systems using Laravel's user authentication component.

By the end of the book, you'll have a complete Laravel application, as well as the tools and knowledge of how to build your own Laravel-based applications unassisted, and where to continue your learning of the framework.
What This Book Covers

Chapter 1, *An Introduction to Laravel*, takes a look at application frameworks in PHP in general, a recent history of the Laravel framework, and the principles that the Laravel framework is built upon.

Chapter 2, *Setting Up a Development Environment*, lays the foundation for what's needed to build Laravel applications by installing and configuring the Homestead virtual machine and the Composer dependency manager.

Chapter 3, *Your First Application*, builds a working application in Laravel from start to finish. Here is where the fun begins!

Chapter 4, *Eloquent ORM*, takes a look at Eloquent, the object relation mapper that ships with Laravel and allows you to query your databases easily.

Chapter 5, *Testing – It's Easier Than You Think*, goes over the various approaches to test your Laravel applications to make sure they're as solid as possible and still work as intended after adding new features.

Chapter 6, *A Command-line Companion Called Artisan*, helps us meet Artisan, the command-line utility for Laravel. We cover the commands Artisan offers out of the box, as well as how to create our own command-line tools.

Chapter 7, *Authentication and Security*, shows you the various ways to protect your Laravel applications from common attacks, as well as how to authenticate and authorize users accessing your application.

Appendix, *An Arsenal of Tools*, covers the arsenal of tools that Laravel provides, which haven't been covered in the previous chapters.
An Introduction to Laravel

PHP frameworks aren’t new, but one of the newest on the block is Laravel. Since version 3, Laravel has exploded in popularity to become one of the most popular and widely used PHP frameworks in a short span of time. At the time of writing, the Laravel repository on GitHub has more stars than its more mature contemporaries such as Symfony, CakePHP, CodeIgniter, and Yii. So what is it about Laravel that makes it so popular?

In this chapter, we will cover the following topics:

• How the productivity can be increased using a framework
• The fundamental concepts and key features of Laravel
• The general structure and conventions of a new Laravel application
• An introduction to the Model-View-Controller (MVC) design pattern, on which Laravel is based
• Migration tips for users of the previous versions of Laravel

We will look at its key features and how they have made Laravel an indispensable tool for many web developers. We will compare writing PHP applications with and without a framework, and see how using a framework can aid in writing more robust and better-structured PHP applications. Then, we will take a closer look at the anatomy of a Laravel application and the third-party packages that it leverages. After reading this chapter, you will have the knowledge needed to get started and build your first Laravel application.
The need for frameworks
Of all the server-side programming languages, PHP undoubtedly has the lowest entry barriers. It is almost always installed by default on even the cheapest web hosts, and it is also extremely easy to set up on any personal computer. For newcomers who have some experience with authoring web pages in HTML and CSS, the concepts of variables, inline conditions, and include statements are easy to grasp. PHP also provides many commonly used functions that one might need when developing a dynamic website. All of this contributes to what some refer to as the immediacy of PHP. However, this instant gratification comes at a cost. It gives a false sense of productivity to beginners, who almost inevitably end up with convoluted spaghetti code as they add more features and functionality to their site. This is mainly because PHP, out of the box, does not do much to encourage the separation of concerns.

The limitations of homemade tools
If you already have a few PHP projects under your belt, but have not used a web application framework before, then you will probably have amassed a personal collection of commonly used functions and classes that you can use on new projects. These homegrown utilities might help you with common tasks, such as sanitizing data, authenticating users, and including pages dynamically. You might also have a predefined directory structure where these classes and the rest of your application code reside. However, all of this will exist in complete isolation; you will be solely responsible for the maintenance, inclusion of new features, and documentation. For a lone developer or an agency with ever-changing staff, this can be a tedious and time-consuming task, not to mention that if you were to collaborate with other developers on the project, they would first have to get acquainted with the way in which you build applications.

Laravel to the rescue
This is exactly where a web application framework such as Laravel comes to the rescue. Laravel reuses and assembles existing components to provide you with a cohesive layer upon which you can build your web applications in a more structured and pragmatic way. Drawing inspiration from popular frameworks written not just in PHP but other programming languages too, Laravel offers a robust set of tools and an application architecture that incorporates many of the best features of frameworks like CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra, and others.

Most of these frameworks use the Model-View-Controller (MVC) paradigm or design pattern. If you have used one of the aforementioned tools or the MVC pattern, then you will find it quite easy to get started with Laravel 5.
A new approach to developing PHP applications

As previously mentioned, PHP gained a bad reputation over the years due to lots of badly-written websites and web applications, and its shortcomings when compared to other, more mature languages. PHP is also notorious for its naming inconsistencies and questionable design decisions regarding its syntax. As a consequence, there has been an exodus to more credible frameworks written in Ruby and Python. Since these languages were nowhere as feature-rich for the Web as PHP, the creators of Ruby on Rails and Django, for instance, had to recreate some essential building blocks, such as classes, to represent HTTP requests and responses and were, therefore, able to avoid some of the mistakes that PHP had made before them, due to the luxury of starting from a blank slate. These frameworks also forced the developer to adhere to a predefined application architecture.

However, it's now a great time to discover (or fall back in love with) PHP again, as over the past couple of years the language has rapidly evolved to include new features such as closures and traits, and a de facto package manager in Composer. Past complaints of PHP when compared to other languages are now exactly that, of the past, and PHP is slowly but surely changing the bad reputation it has suffered from, for so long.

A more robust HTTP foundation

After years of people developing their own, unique approach of handling common tasks, such as handling requests and responses, specifically for their own projects, one framework took a different approach and instead, began creating components that could be used in any codebase no matter its foundation, be it homegrown or based on a framework. The Symfony project adopted these principles to recreate a more solid, flexible, and testable HTTP foundation for PHP applications. Along with the latest version of Drupal and phpBB, Laravel is one of the many open source projects that use this foundation together with several other components that form the Symfony framework.

Laravel is such a project that relies on the HTTP foundation created by Symfony. It also relies on other components created by Symfony, as well as a variety of other popular libraries, such as SwiftMailer for more straightforward e-mailing, Carbon for more expressive date and time handling, Doctrine for its inflector and database abstraction tools, and a handful of other tools to handle logging, class loading, and error reporting. Instead of re-inventing the wheel, Laravel decided to hop on the shoulder of giants and embrace these pre-existing mature components.
Embracing PHP

One way in which Laravel differs from its contemporaries is that it openly embraces new features of PHP and in turn requires a fairly recent version (at least 5.4). Previously, other frameworks would build support for older versions of PHP to maintain backwards-compatibility for as long as possible. However, this approach meant that those same frameworks couldn't take advantage of new features in the newer versions of PHP, in turn, hampering the evolution of PHP. Using Laravel 5, you will get to grips with some of the newer features of PHP. If you're new to PHP, or coming back to the language after a while, then here's what you can expect to find:

- **Namespaces**: More mature languages such as Java and C# have namespaces. Namespaces help developers avoid naming collisions that might happen if say, two different libraries have the same function or class name. In PHP, namespaces are separated by backslashes, which is usually mirrored by the directory structure, with the only difference being the use of slashes on Unix systems, in accordance with the PSR-4 convention. A namespace, such as `namespace Illuminate\Database\Eloquent` is declared at the top of the file. To use code from another namespace, it needs to be imported, which can be done with the `use` keyword, and then by specifying the namespace, that is, `use Illuminate\Database\Eloquent\Model`. Another advantage of namespaces is that you can alias imported classes, so as to avoid collisions with classes with the same name in another namespace or the global namespace. To do this, you use the `as` keyword after the `use` statement as `use Foo\Logger as FooLogger;`

- **Interfaces**: Interfaces specify the methods that a class should provide when that interface is implemented. Interfaces do not contain any implementation details themselves, merely the methods (and the arguments those methods should take). For instance, if a class implements Laravel's `JsonableInterface` instance, then that class will also need to have a `toJson()` method. Within Laravel, interfaces tend to be referred to as **Contracts**.

- **Anonymous functions**: These are also known as closures and were introduced in PHP 5.3. Somewhat reminiscent of JavaScript, they help you to produce shorter code, and you will use them extensively when building Laravel applications to define routes, events, filters, and in many other instances. This is an example of an anonymous function attached to a route: `Route::get('/', function() { return 'Hello, world.'; });`. In Laravel, this code creates a new route when the base path of a website is requested. When it is, the code in the closure is executed and returned as the response.
• **Overloading**: Also called dynamic methods, they allow you to call methods such as `whereUsernameOrEmail($name, $email)` that were not explicitly defined in a class. These calls get handled by the `__call()` method in the class, which then tries to parse the name to execute one or more known methods. In this case, `->where('username', $username)->orWhere('email', $email).

• **Shorter array syntax**: PHP 5.4 introduced the shorter array syntax. Instead of writing `array('primes' =>array(1,3,5,7))`, it is now possible to use just square brackets to denote an array, that is, `['primes'=>[1,3,5,7]]`. You might know syntax if you've used arrays in JavaScript.

**Laravel's main features and sources of inspiration**

So, what do you get out of the box with Laravel 5? Let's take a look and see how the following features can help boost your productivity:

• **Modularity**: Laravel was built on top of over 20 different libraries and is itself split into individual modules. Tightly integrated with Composer dependency manager, these components can be updated with ease.

• **Testability**: Built from the ground up to ease testing, Laravel ships with several helpers that let you visit routes from your tests, crawl the resulting HTML, ensure that methods are called on certain classes, and even impersonate authenticated users in order to make sure the right code is run at the right time.

• **Routing**: Laravel gives you a lot of flexibility when you define the routes of your application. For example, you could manually bind a simple anonymous function to a route with an HTTP verb, such as `GET`, `POST`, `PUT`, or `DELETE`. This feature is inspired by micro-frameworks, such as Sinatra (Ruby) and Silex (PHP).

• **Configuration management**: More often than not, your application will be running in different environments, which means that the database or e-mail server credential's settings or the displaying of error messages will be different when your app is running on a local development server to when it is running on a production server. Laravel has a consistent approach to handle configuration settings, and different settings can be applied in different environments via the use of an `.env` file, containing settings unique for that environment.
An Introduction to Laravel

- **Query builder and ORM**: Laravel ships with a fluent query builder, which lets you issue database queries with a PHP syntax, where you simply chain methods instead of writing SQL. In addition to this, it provides you with an Object Relational Mapper (ORM) and ActiveRecord implementation, called Eloquent, which is similar to what you will find in Ruby on Rails, to help you define interconnected models. Both the query builder and the ORM are compatible with different databases, such as PostgreSQL, SQLite, MySQL, and SQL Server.

- **Schema builder, migrations, and seeding**: Also inspired by Rails, these features allow you to define your database schema in PHP code and keep track of any changes with the help of database migrations. A migration is a simple way of describing a schema change and how to revert to it. Seeding allows you to populate the selected tables of your database, for example, after running a migration.

- **Template engine**: Partly inspired by the Razor template language in ASP.NET MVC, Laravel ships with Blade, a lightweight template language with which you can create hierarchical layouts with predefined blocks in which dynamic content is injected.

- **E-mailing**: With its Mail class, which wraps the popular SwiftMailer library, Laravel makes it very easy to send an e-mail, even with rich content and attachments from your application. Laravel also comes with drivers for popular e-mail sending services such as SendGrid, Mailgun, and Mandrill.

- **Authentication**: Since user authentication is such a common feature in web applications, out of the box Laravel comes with a default implementation to register, authenticate, and even send password reminders to users.

- **Redis**: This is an in-memory key-value store that has a reputation for being extremely fast. If you give Laravel a Redis instance that it can connect to, it can use it as a session and general purpose cache, and also give you the possibility to interact with it directly.

- **Queues**: Laravel integrates with several queue services, such as Amazon SQS, Beanstalkd, and IronMQ, to allow you to delay resource-intensive tasks, such as the e-mailing of a large number of users, and run them in the background, rather than keep the user waiting for the task to complete.

- **Event and command bus**: Although not new in version 5, Laravel has brought a command bus to the forefront in which it's easy to dispatch events (a class that represents something that's happened in your application), handle commands (another class that represents something that should happen in your application), and act upon these at different points in your application's lifecycle.
Expressiveness and simplicity

Something that is at the core of Laravel is its philosophy that code should be named simply and expressively. Consider the following code example:

```php
<?php
Route::get('area/{area}', function($area) {
    if (51 == $area && ! Auth::check()) {
        return Redirect::guest('login');
    } else {
        return 'Welcome to Area '.$area;
    }
})-where('area, '[0-9]+');
```

Even though we have not even touched Laravel or covered its routing functions yet, you will probably have a rough idea of what this snippet of code does. Expressive code is more readable for someone new to a project, and it is probably also easier for you to learn and remember.

Prettifying PHP

Prettifying PHP as well as ensuring code in Laravel is named to effectively convey its actions in plain English, the authors of Laravel have also gone on to apply these principles to existing PHP language functions. A prime example is the `Storage` class, which was created to make file manipulations:

- **More expressive**: To find out when a file was last modified, use `Storage::lastModified($path)` instead of `filemtime(realpath($path))`. To delete a file, use `Storage::delete($path)` instead of `unlink($path)`, which is the plain old PHP equivalent.

- **More consistent**: Some of the original file manipulation functions of PHP are prefixed with `file_`, while others just start with `file`; some are abbreviated and other are not. Using Laravel’s wrappers, you no longer need to guess or refer to PHP’s documentation.
• More testable: Many of the original functions can be tricky to use in tests, due to the exceptions they throw and also because they are more difficult to mock.

• More feature complete: This is achieved by adding functions that did not exist before, such as `File::copyDirectory($directory, $destination)`.

There are very rare instances where expressiveness is foregone in the favor of brevity. This is the case for commonly-used shortcut functions, such as `e()`, that escape HTML entities, or `dd()`, with which you can halt the execution of the script and dump the contents of one or more variables.

Responsibilities, naming, and conventions

At the beginning of this chapter, we pointed out that one of the main issues with standard PHP applications was the lack of a clear separation of concerns; business logic becomes entangled with the presentation and data tier. Like many other frameworks that favor convention over configuration, Laravel gives you scaffolding with predefined places to put code in. To help you eliminate trivial decisions, it expects you to name your variables, methods, or database tables in certain ways, even though these are editable via configuration. It is, however, far less opinionated than a framework such as Ruby on Rails and in areas like routing, where there is often more than one way to solve a problem.

You might remember us mentioning that Laravel is a framework that is based on the MVC paradigm. Do not worry if you have not used this architectural pattern before; in a nutshell, this is what you need to know about MVC in order to be able to build your first Laravel applications:

• Models: Models represent resources in your application. More often than not, they correspond to records in a data store, most commonly a database table. In this respect, you can think of models as representing entities in your application, be that a user, a news article, or an event, among others. In Laravel, models are classes that usually extend Eloquent's base Model class and are named in CamelCase (that is, `NewsArticle`). This will correspond to a database table with the same name, but in snake_case and plural (that is, `news_articles`). By default, Eloquent also expects a primary key named `id`, and will also look for—and automatically update—the `created_at` and `updated_at` columns. Models can also describe the relationships they have with other models. For example, a `NewsArticle` model might be associated with a `User` model, as a `User` model might be able to author a `NewsArticle` model. However, models can also refer to data from other data sources, such as an XML file, or the response from a web service or API.
• **Controllers or routes**: Controllers, at their simplest, take a request, do something, and then send an appropriate response. Controllers are where the actual processing of data goes, whether that is retrieving data from a database, or handling a form submission, and saving data back to a database. Although you are not forced to adhere to any rules when it comes to creating controller classes in Laravel, it does offer you two sane approaches: RESTful controllers and resource controllers. A RESTful controller allows you to define your own actions and what HTTP methods they should respond to. Resource controllers are based around an entity and allow you to perform common operations on that entity, based on the HTTP method used. Another option is to bypass using controller classes altogether and instead write your logic in your routes, by way of anonymous functions.

• **Views or Templates**: Views are responsible for displaying the response returned from a controller in a suitable format, usually as an HTML web page. They can be conveniently built by using the Blade template language or by simply using standard PHP. The file extension of the view, either .blade.php or simply .php, determines whether or not Laravel treats your view as a Blade template or not.

The following diagram illustrates the interactions between all the constituents applied in a typical web application:
Of course, it is possible to go against the MVC paradigm and the framework's conventions and write code as you wish, but this will often require more effort on the developer's part for no gain.

**Helping you become a better developer**

Laravel has become a standard-bearer for a new way of developing PHP applications through various design decisions and philosophies, such as the way in which it advocates developers to write framework-agnostic code and to rely on contracts (interfaces) rather than implementations are only a good thing. It has also built such a strong community that it is undoubtedly one of its strongest assets and a major contributing factor to its success; it is possible to get answers within minutes from other users via avenues such as forums, IRC, and social networking websites like Twitter.

However, if time has taught us anything, it is that frameworks come and go and it is hard to predict when Laravel will lose its steam and be supplanted by a better or more popular framework. Nonetheless, Laravel will not only make you more productive in the short term, but it also has the potential to make you a better developer in the long run. By using it to build web applications, you will indirectly become more familiar with the following concepts, all of which are highly transferable to any other programming language or framework. These include the MVC paradigm and **Object-oriented programming (OOP)** design patterns, the use of dependency managers, testing and dependency injection, and the power and limitations of ORMs and database migration.

It will also inspire you to write more expressive code with descriptive **DocBlock** comments that facilitate the generation of documentation, as well as the future maintenance of the application, irrespective of whether it is done by you or another developer.

**Structure of a Laravel application**

Over the course of the next two chapters, we will install Laravel and create our first application. Like most frameworks, Laravel starts out with a complete directory tree for you to organize your code in, and also includes placeholder files for you to use as a starting point. Here is what the directory of a new Laravel 5 application looks like:

```
./app/ # Your Laravel application
   ./app/Commands/ # Commands classes
      ./app/Console/Commands/ # Command-line scripts
   ./app/Events/ raise # Events that your application can
   ./app/Exceptions/
```
Chapter 1

./app/Handlers/ # Exception handlers
./app/Handlers/Commands # Handlers for command classes
./app/Handlers/Events # Handlers for event classes
./app/Http/ # Your application's controllers
./app/Http/Middleware/ # Filters applied to requests
./app/Http/Requests/ # Classes that can modify requests
./app/Http/routes.php # URLs and their corresponding handlers
./app/Providers # Service provider classes
./app/Services # Services used in your application
./bootstrap/ # Application bootstrapping scripts
./config/ # Configuration files
./database/ # Database migration classes
./database/migrations/ # Database seeder classes
./database/seeds/ # Your application's document root
./public/ # Sends incoming requests to index.php
./public/.htaccess # Starts Laravel application
./public/index.php
./resources/ # Hold raw assets like LESS & Sass files
./resources/assets/ # Localization and language files
./resources/lang/ # Templates that are rendered as HTML
./resources/views/
./storage/ # App storage, like file uploads etc
./storage/app/ # Framework storage (cache)
./storage/framework/ # Contains application-generated logs
./storage/logs/
./tests/ # Test cases
./vendor/ # Third-party code installed by Composer
./.env.example # Example environment variable file
./artisan # Artisan command-line utility
./composer.json # Project dependencies manifest
./phpunit.xml # Configures PHPUnit for running tests
./server.php # A lightweight local development server
Like Laravel’s source code, the naming of directories is also expressive, and it is easy to guess what each directory is for. The app directory is where most of your application’s server-side code will reside, which has subdirectories both for how your application could be accessed (Console and Http), as well as subdirectories for organizing code that could be used in both scenarios (such as Events and Services). We will explore the responsibilities of each directory further in the next chapters.

The service container and request lifecycle

Whether you are a beginner in PHP or an experienced developer in a different language, it might not always be obvious how an HTTP request reaches a Laravel application. Indeed, the request lifecycle is fundamentally different from plain PHP scripts that are accessed directly by their URI (for example, GET http://example.com/about-us.php).

The public/ directory is meant to act as the document root; in other words, the directory in which your web server starts looking after every incoming request. Once URL rewriting is properly set up, every request that does not match an existing file or directory hits the /public/index.php file. This file includes the Composer autoloader file, which loads in dependencies (including the Laravel framework components) and also where to look for your application’s code. Your application is then bootstrapped, loading configuration variables based on the environment. Once this is done, it instantiates a new service container instance, which in turn handles the incoming request, uses the HTTP method and URL used to access the application (such as POST /comments), and passes the request off to the correct controller action or route for handling.

Exploring Laravel

In this chapter, we are only covering the general mechanisms of how Laravel works, without looking at the detailed implementation examples. For the majority of developers, who just want to get the job done, this is sufficient. Moreover, it is much easier to delve into the source code of Laravel once you have already built a few applications. Nevertheless, here are some answers to the questions that might crop up when exceptions are thrown or when you navigate through the source code. In doing so, you will come across some methods that are not documented in the official guide, and you might even be inspired to write better code.
Browsing the API (http://laravel.com/api) can be somewhat intimidating at first. But it is often the best way to understand how a particular method works under the hood. Here are a few tips:

- The Illuminate namespace does not refer to a third-party library. It is the namespace that the author of Laravel has chosen for the different modules that constitute Laravel. Every single one of them is meant to be reusable and used independently of the framework.

- When searching for a class definition, for example, Auth, in the source code or the API, you might bump into Facade, which hardly contains any helpful methods and only acts as a proxy to the real class. This is because almost every dependency in Laravel is injected into the service container when it is instantiated.

- Most of the libraries that are included in the vendor/ directory contain a README file, which details the functionality present in the library (for example, vendor/nesbot/carbon/readme.md).

**Changes in Version 5 from Version 4**

Laravel 5 started life as Laravel 4.3, but was promoted to its own major version when it became apparent that this new version was going to be a radical departure from version 4 of the framework. Laravel 5 builds on Laravel 4 as a base, but makes architecting larger applications with things like an application namespace out of the box. Laravel 4 applications will need a fair bit of work to be ported to Laravel 5. Features that are new or have been updated in Laravel 5 include:

- **Method injection:** In Laravel 4, you could type hint (specify in the constructor) the dependencies a class needed, and Laravel would automatically resolve those dependencies out of its container. Now, Laravel 5 takes that one step further and will also resolve dependencies specified in class methods, as well as class constructors.

- **Form requests:** Laravel 5 introduces form request classes. These classes can be injected into your controller actions. They take the current request, and on it, you can perform data validation and sanitizing and even user authorization (that is, check if the currently-logged-in user can perform the requested action). This streamlines validation, meaning you have to do very little, if any, data validation in your controller actions.

- **Socialite:** New to Laravel 5 is an optional package called Socialite that you can declare as a Composer dependency. It makes authenticating with third-party services a breeze, meaning you can easily implement functionality like login with Facebook in a few lines of code.
• **Elixir**: Laravel 5 also looks at making front-end development easier. A lot of developers these days are using languages like LESS and Sass to create their style sheets, and concatenating JavaScript files into one, minified JavaScript file to reduce HTTP requests and speed up loading times. Elixir is a wrapper around [Gulp](https://github.com/gulpjs/gulp), a [Node.js](https://nodejs.org) based build system that simplifies the tasks mentioned here. This greatly reduces the time needed to get up and running with a new application, as you don't have to install Node.js modules or Gulp files from other projects. You get it free from the get-go.

**Summary**

In this chapter, we have introduced you to Laravel 5 and how it can help you to write better, more structured applications while reducing the amount of boilerplate code. We have also explained the concepts and PHP features used by Laravel, and you should now be well equipped to get started and write your first application!

In the next chapter, you will learn how to set up an environment in which you can develop Laravel applications and you will also be introduced to Composer for managing dependencies.
Where to buy this book

You can buy Laravel 5 Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.