



# Laravel Up & Running

---

A FRAMEWORK FOR BUILDING MODERN PHP APPS

Matt Stauffer

# Laravel: Up and Running

What sets Laravel apart from other PHP web frameworks? Speed and simplicity, for starters. This rapid application development framework and its vast ecosystem of tools let you quickly build new sites and applications with clean, readable code. With this practical guide, Matt Stauffer—a leading teacher and developer in the Laravel community—provides the definitive introduction to one of today's most popular web frameworks.

The book's high-level overview and concrete examples will help experienced PHP web developers get started with Laravel right away. By the time you reach the last page, you should feel comfortable writing an entire application in Laravel from scratch.

Dive into several features of this framework, including:

- Blade, Laravel's powerful, custom templating tool
- Tools for gathering, validating, normalizing, and filtering user-provided data
- Laravel's Eloquent ORM for working with the application's databases
- The Illuminate request object, and its role in the application lifecycle
- PHPUnit and Mockery for testing your PHP code
- Laravel's tools for writing JSON and RESTful APIs
- Interfaces for file system access, sessions, cookies, caches, and search
- Tools for implementing queues, jobs, events, and WebSocket event publishing
- Laravel's specialty packages: Scout, Passport, Cashier, Echo, Elixir, Valet, and Socialite

---

**Matt Stauffer** is a developer and a teacher. He is a partner and technical director at Tighten Co., blogs at [mattstauffer.co](http://mattstauffer.co), and hosts The Five-Minute Geek Show and the Laravel Podcast.

---

PHP

US \$44.99

CAN \$51.99

ISBN: 978-1-491-93608-5



9 781491 936085



“Not only is Matt one of the most knowledgeable members of the Laravel community, he is also a fantastic teacher. I'm proud to recommend this book as a thorough, extensive guide to the Laravel framework.”

—Taylor Otwell  
Creator of Laravel

“As a self-diagnosed evangelist for Laravel, I'm thankful to finally have a book that I can point newcomers to. Even better, I can think of no one more qualified to introduce this wonderful framework to the masses than Matt.”

—Jeffrey Way  
Founder of Laracasts

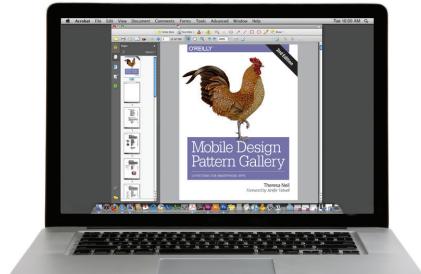
Twitter: @oreillymedia  
[facebook.com/oreilly](http://facebook.com/oreilly)

# O'Reilly ebooks.

Your bookshelf on your devices.



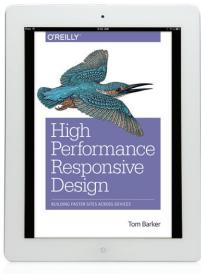
PDF



Mobi



ePub



DAISY

When you buy an ebook through [oreilly.com](http://oreilly.com) you get lifetime access to the book, and whenever possible we provide it to you in four DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

**Learn more at [ebooks.oreilly.com](http://ebooks.oreilly.com)**

You can also purchase O'Reilly ebooks through the iBookstore, the [Android Marketplace](http://Android Marketplace), and [Amazon.com](http://Amazon.com).

**O'REILLY®**

## **Laravel: Up and Running**

by Matt Stauffer

Copyright © 2017 Matt Stauffer. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com/safari>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Allyson MacDonald

**Production Editor:** Colleen Lobner

**Copieditor:** Rachel Head

**Proofreader:** Kim Cofer

**Indexer:** Angela Howard

**Interior Designer:** David Futato

**Cover Designer:** Randy Comer

**Illustrator:** Rebecca Demarest

December 2016: First Edition

### **Revision History for the First Edition**

2016-11-14: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781491936085> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Laravel: Up and Running*, the cover image of a gemsbok, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-93608-5

[LSI]

---

# Table of Contents

<b>Preface.....</b>	<b>xv</b>
<b>1. Why Laravel?.....</b>	<b>1</b>
Why Use a Framework?	1
“I’ll Just Build It Myself”	2
Consistency and Flexibility	2
A Short History of Web and PHP Frameworks	2
Ruby on Rails	2
The Influx of PHP Frameworks	3
The Good and the Bad of CodeIgniter	3
Laravel 1, 2, and 3	3
Laravel 4	4
Laravel 5	4
What’s So Special About Laravel?	4
The Philosophy of Laravel	4
How Laravel Achieves Developer Happiness	5
The Laravel Community	6
How It Works	7
Why Laravel?	9
<b>2. Setting Up a Laravel Development Environment.....</b>	<b>11</b>
System Requirements	11
Composer	12
Local Development Environments	12
Laravel Valet	12
Laravel Homestead	13
Creating a New Laravel Project	17
Installing Laravel with the Laravel Installer Tool	17

Installing Laravel with Composer's create-project Feature	17
Laravel's Directory Structure	18
The Folders	18
The Loose Files	19
Configuration	19
Up and Running	20
Testing	21
TL;DR	21
<b>3. Routing and Controllers.....</b>	<b>23</b>
Route Definitions	23
Route Verbs	25
Route Handling	26
Route Parameters	26
Route Names	28
Route Groups	30
Middleware	31
Path Prefixes	32
Subdomain Routing	33
Namespace Prefixes	33
Name Prefixes	34
Views	34
Using View Composers to Share Variables with Every View	35
Controllers	36
Getting User Input	39
Injecting Dependencies into Controllers	40
Resource Controllers	41
Route Model Binding	43
Implicit Route Model Binding	43
Custom Route Model Binding	44
Route Caching	45
Form Method Spoofing	45
An Introduction to HTTP Verbs	45
HTTP Verbs in Laravel	46
HTTP Method Spoofing in HTML Forms	46
CSRF Protection	46
Redirects	48
<code>redirect()-&gt;to()</code>	49
<code>redirect()-&gt;route()</code>	49
<code>redirect()-&gt;back()</code>	50
Other Redirect Methods	50
<code>redirect()-&gt;with()</code>	50

Abort the Request	52
Custom Responses	52
<code>response()-&gt;make()</code>	53
<code>response()-&gt;json()</code> and <code>-&gt;jsonp()</code>	53
<code>response()-&gt;download()</code> and <code>-&gt;file()</code>	53
Testing	53
TL;DR	54
<b>4. Blade Templating.....</b>	<b>55</b>
Echoing Data	56
Control Structures	57
Conditionals	57
Loops	58
or	60
Template Inheritance	60
Defining Sections with <code>@section/@show</code> and <code>@yield</code>	60
<code>@parent</code>	62
<code>@include</code>	62
<code>@each</code>	63
View Composers and Service Injection	64
Binding Data to Views Using View Composers	64
Blade Service Injection	67
Custom Blade Directives	68
Parameters in Custom Blade Directives	70
Example: Using Custom Blade Directives for a Multitenant App	70
Testing	71
TL;DR	72
<b>5. Frontend Components.....</b>	<b>75</b>
Elixir	75
Elixir Folder Structure	77
Running Elixir	77
What Does Elixir Provide?	78
Pagination	84
Paging Database Results	84
Manually Creating Paginators	85
Message Bags	86
Named Error Bags	88
String Helpers, Pluralization, and Localization	88
The String Helpers and Pluralization	88
Localization	89
Testing	92

Testing with Elixir	92
Testing Message and Error Bags	92
Translation and Localization	92
TL;DR	93
<b>6. Collecting and Handling User Data.....</b>	<b>95</b>
Injecting a Request Object	95
\$request->all()	96
\$request->except() and \$request->only()	96
\$request->has() and \$request->exists()	97
\$request->input()	97
Array Input	98
JSON Input (and \$request->json())	98
Route Data	100
From Request	100
From Route Parameters	100
Uploaded Files	101
Validation	103
validate() in the Controller Using ValidatesRequests	103
Manual Validation	106
Displaying Validation Error Messages	106
Form Requests	107
Creating a Form Request	107
Using a Form Request	108
Eloquent Model Mass Assignment	109
{{ Versus>{!!	110
Testing	110
TL;DR	111
<b>7. Artisan and Tinker.....</b>	<b>113</b>
An Introduction to Artisan	113
Basic Artisan Commands	114
Options	114
The Grouped Commands	115
Writing Custom Artisan Commands	117
Registering Commands	119
A Sample Command	120
Arguments and Options	121
Using Input	123
Prompts	124
Output	125
Calling Artisan Commands in Normal Code	127

Tinker	128
Testing	128
TL;DR	129
<b>8. Database and Eloquent.....</b>	<b>131</b>
Configuration	131
Database Connections	132
Other Database Configuration Options	133
Migrations	133
Defining Migrations	134
Running Migrations	141
Seeding	141
Creating a Seeder	142
Model Factories	143
Query Builder	146
Basic Usage of the DB Facade	147
Raw SQL	147
Chaining with the Query Builder	149
Transactions	156
Introduction to Eloquent	157
Creating and Defining Eloquent Models	159
Retrieving Data with Eloquent	160
Inserts and Updates with Eloquent	162
Deleting with Eloquent	165
Scopes	168
Customizing Field Interactions with Accessors, Mutators, and Attribute Casting	171
Eloquent Collections	174
Eloquent Serialization	177
Eloquent Relationships	179
Child Records Updating Parent Record Timestamps	190
Eloquent Events	192
Testing	193
TL;DR	195
<b>9. User Authentication and Authorization.....</b>	<b>197</b>
The User Model and Migration	198
Using the auth() Global Helper and the Auth Facade	201
The Auth Controllers	201
RegisterController	201
LoginController	203
ResetPasswordController	204

ForgotPasswordController	205
Auth::routes()	205
The Auth Scaffold	206
“Remember Me”	207
Manually Authenticating Users	208
Auth Middleware	208
Guards	209
Changing the Default Guard	209
Using Other Guards Without Changing the Default	210
Adding a New Guard	210
Creating a Custom User Provider	211
Custom User Providers for Nonrelational Databases	211
Auth Events	212
Authorization (ACL) and Roles	212
Defining Authorization Rules	213
The Gate Facade (and Injecting Gate)	214
The Authorize Middleware	214
Controller Authorization	215
Checking on the User Instance	217
Blade Checks	217
Intercepting Checks	218
Policies	218
Testing	221
TL;DR	223
<b>10. Requests and Responses.....</b>	<b>225</b>
Laravel’s Request Lifecycle	225
Bootstrapping the Application	226
Service Providers	227
The Request Object	228
Getting a Request Object in Laravel	229
Getting Basic Information About a Request	230
Persistence	233
The Response Object	233
Using and Creating Response Objects in Controllers	233
Specialized Response Types	235
Laravel and Middleware	238
An Introduction to Middleware	238
Creating Custom Middleware	239
Binding Middleware	241
Passing Parameters to Middleware	244
Testing	245

TL;DR	246
<b>11. The Container.....</b>	<b>247</b>
A Quick Introduction to Dependency Injection	247
Dependency Injection and Laravel	249
The <code>app()</code> Global Helper	249
How the Container Is Wired	250
Binding Classes to the Container	251
Binding to a Closure	252
Binding to Singletons, Aliases, and Instances	253
Binding a Concrete Instance to an Interface	253
Contextual Binding	254
Constructor Injection	254
Method Injection	255
Facades and the Container	256
How Facades Work	257
Service Providers	258
Testing	258
TL;DR	259
<b>12. Testing.....</b>	<b>261</b>
Testing Basics	262
Naming Tests	264
The Testing Environment	265
The Testing Traits	266
WithoutMiddleware	266
DatabaseMigrations	266
DatabaseTransactions	266
Application Testing	266
TestCase	267
“Visiting” Routes	268
Custom Application Testing Assertions	270
JSON and Non-visit() Application Testing Assertions	271
Clicking and Forms	275
Jobs and Events	276
Authentication and Sessions	277
Artisan and Seed	277
Mocking	278
Mockery	278
Mocking Facades	281
TL;DR	282

<b>13. Writing APIs.....</b>	<b>283</b>
The Basics of REST-Like JSON APIs	283
Controller Organization and JSON Returns	285
Reading and Sending Headers	288
Sending Response Headers in Laravel	289
Reading Request Headers in Laravel	289
Eloquent Pagination	289
Sorting and Filtering	291
Sorting Your API Results	291
Filtering Your API Results	293
Transforming Results	293
Writing Your Own Transformer	294
Nesting and Relationships	295
API Authentication with Laravel Passport	297
A Brief Introduction to OAuth 2.0	297
Installing Passport	297
Passport's API	299
Passport's Available Grant Types	299
Managing Clients and Tokens with the Passport API and the Vue Components	307
Passport Scopes	309
Laravel 5.2+ API Token Authentication	311
Testing	312
TL;DR	313
<b>14. Storage and Retrieval.....</b>	<b>315</b>
Local and Cloud File Managers	315
Configuring File Access	315
Using the Storage Facade	316
Adding Additional Flysystem Providers	318
Basic File Uploads and Manipulation	318
Sessions	320
Accessing the Session	320
The Methods Available on Session Instances	321
Flash Session Storage	322
Cache	323
Accessing the Cache	323
The Methods Available on Cache Instances	324
Cookies	325
Cookies in Laravel	325
Accessing the Cookie Tools	326
Full-Text Search with Laravel Scout	329

Installing Scout	329
Marking Your Model for Indexing	329
Searching Your Index	329
Queues and Scout	330
Perform Operations Without Indexing	330
Manually Trigger Indexing via Code	330
Manually Trigger Indexing via the CLI	331
Testing	331
File Storage	331
Session	333
Cache	334
Cookies	334
TL;DR	335
<b>15. Mail and Notifications.....</b>	<b>337</b>
Mail	337
“Classic” Mail	338
Basic “Mailable” Mail Usage	338
Mail Templates	340
Methods Available in build()	341
Attachments and Inline Images	342
Queues	343
Local Development	344
Notifications	345
Defining the via() Method for Your Notifiables	348
Sending Notifications	349
Queueing Notifications	349
Out-of-the-Box Notification Types	350
Testing	352
Mail	352
Notifications	353
TL;DR	354
<b>16. Queues, Jobs, Events, Broadcasting, and the Scheduler.....</b>	<b>355</b>
Queues	355
Why Queues?	356
Basic Queue Configuration	356
Queued Jobs	356
Running a Queue Worker	360
Handling Errors	360
Controlling the Queue	363
Queues Supporting Other Functions	364

Events	364
Firing an Event	364
Listening for an Event	366
Broadcasting Events over WebSockets, and Laravel Echo	369
Configuration and Setup	370
Broadcasting an Event	370
Receiving the Message	373
Advanced Broadcasting Tools	374
Laravel Echo (the JavaScript Side)	378
Scheduler	382
Available Task Types	383
Available Time Frames	383
Blocking and Overlap	385
Handling Task Output	385
Task Hooks	386
Testing	386
TL;DR	388
<b>17. Helpers and Collections</b> .....	<b>389</b>
Helpers	389
Arrays	389
Strings	391
Application Paths	393
URLs	394
Misc	395
Collections	398
The Basics of Collections	399
A Few Methods	400
TL;DR	405
<b>Glossary</b> .....	<b>407</b>
<b>Index</b> .....	<b>413</b>

# Why Laravel?

In the early days of the dynamic web, writing a web application looked a lot different than it does today. Developers then were responsible for writing the code for not just the unique business logic of our applications, but also each of the components that are so common across sites—user authentication, input validation, database access, templating, and more.

Today, programmers have dozens of application development frameworks and thousands of components and libraries easily accessible. It's a common refrain among programmers that, by the time you learn one framework, three newer (and purportedly better) frameworks have popped up intending to replace it.

“Just because it’s there” might be a valid justification for climbing a mountain, but there are better reasons to choose to use a specific framework—or to use a framework at all. It’s worth asking the question: why frameworks? More specifically, why Laravel?

## Why Use a Framework?

It’s easy to see why it’s beneficial to use the individual components, or packages, that are available to PHP developers. With packages, someone else is responsible for developing and maintaining an isolated piece of code that has a well-defined job, and in theory that person has a deeper understanding of this single component than you have time to have.

Frameworks like Laravel—and Symfony, Silex, Lumen, and Slim—prepackage a collection of third-party components together with custom framework “glue” like configuration files, service providers, prescribed directory structures, and application bootstraps. So, the benefit of using a framework in general is that someone has made decisions not just about individual components for you, but also about *how those components should fit together*.

## “I’ll Just Build It Myself”

Let’s say you start a new web app without the benefit of a framework. Where do you begin? Well, it should probably route HTTP requests, so you now need to evaluate all of the HTTP request and response libraries available and pick one. Then a router. Oh, and you’ll probably need to set up some form of routes configuration file. What syntax should it use? Where should it go? What about controllers? Where do they live, and how are they loaded? Well, you probably need a dependency injection container to resolve the controllers and their dependencies. But which one?

Furthermore, what if you do take the time to answer all those questions and successfully create your application—what’s the impact on the next developer? What about when you have four such custom-framework-based applications, or a dozen, and you have to remember where the controllers live in each, or what the routing syntax is?

## Consistency and Flexibility

Frameworks address this issue by providing a carefully considered answer to the question “Which component should we use here?” and ensuring that the particular components chosen work well together. Additionally, frameworks provide conventions that reduce the amount of code a developer new to the project has to understand—if you understand how routing works in one Laravel project, for example, you understand how it works in all Laravel projects.

When someone prescribes rolling your own framework for each new project, what they’re really advocating is the ability to *control* what does and doesn’t go into your application’s foundation. That means the best frameworks will not only provide you with a solid foundation, but also give you the freedom to customize to your heart’s content. And this, as I’ll show you in the rest of this book, is part of what makes Laravel so special.

## A Short History of Web and PHP Frameworks

An important part of being able to answer the question “Why Laravel?” is understanding Laravel’s history—and understanding what came before it. Prior to Laravel’s rise in popularity, there were a variety of frameworks and other movements in PHP and other web development spaces.

### Ruby on Rails

David Heinemeier Hansson released the first version of Ruby on Rails in 2004, and it’s been hard to find a web application framework since then that hasn’t been influenced by Rails in some way.

Rails popularized MVC, RESTful JSON APIs, convention over configuration, Active-Record, and many more tools and conventions that had a profound influence on the way web developers approached their applications—especially with regard to rapid application development.

## The Influx of PHP Frameworks

It was clear to most developers that Rails, and similar web application frameworks, were the wave of the future, and PHP frameworks, including those admittedly imitating Rails, starting popping up quickly.

CakePHP was the first in 2005, and it was soon followed by Symfony, CodeIgniter, Zend Framework, and Kohana (a CodeIgniter fork). Yii arrived in 2008, and Aura and Slim in 2010. 2011 brought FuelPHP and Laravel, both of which were not quite CodeIgniter offshoots, but instead proposed as alternatives.

Some of these frameworks were more Rails-y, focusing on database object-relational mappers (ORMs), MVC structures, and other tools targeting rapid development. Others, like Symfony and Zend, focused more on enterprise design patterns and ecommerce.

## The Good and the Bad of CodeIgniter

CakePHP and CodeIgniter were the two early PHP frameworks that were most open about how much their inspiration was drawn from Rails. CodeIgniter quickly rose to fame and by 2010 was arguably the most popular of the independent PHP frameworks.

CodeIgniter was simple, easy to use, and boasted amazing documentation and a strong community. But its use of modern technology and patterns advanced slowly, and as the framework world grew and PHP's tooling advanced, CodeIgniter started falling behind in terms of both technological advances and out-of-the-box features. Unlike many other frameworks, CodeIgniter was managed by a company, and they were slow to catch up with PHP 5.3's newer features like namespaces and the moves to GitHub and later Composer. It was in 2010 that Taylor Otwell, Laravel's creator, became dissatisfied enough with CodeIgniter that he set off to write his own framework.

## Laravel 1, 2, and 3

The first beta of Laravel 1 was released in June 2011, and it was written completely from scratch. It featured a custom ORM (Eloquent); closure-based routing (inspired by Ruby Sinatra); a module system for extension; and helpers for forms, validation, authentication, and more.

Early Laravel development moved quickly, and Laravel 2 and 3 were released in November 2011 and February 2012, respectively. They introduced controllers, unit testing, a command-line tool, an inversion of control (IoC) container, Eloquent relationships, and migrations.

## Laravel 4

With Laravel 4, Taylor rewrote the entire framework from the ground up. By this point Composer, PHP's now-ubiquitous package manager, was showing signs of becoming an industry standard and Taylor saw the value of rewriting the framework as a collection of components, distributed and bundled together by Composer.

Taylor developed a set of components under the code name *Illuminate* and, in May 2013, released Laravel 4 with an entirely new structure. Instead of bundling the majority of its code as a download, Laravel now pulled in the majority of its components from Symfony (another framework that released its components for use by others) and the Illuminate components through Composer.

Laravel 4 also introduced queues, a mail component, facades, and database seeding. And because Laravel was now relying on Symfony components, it was announced that Laravel would be mirroring (not exactly, but soon after) the six-monthly release schedule Symfony follows.

## Laravel 5

Laravel 4.3 was scheduled to release in November 2014, but as development progressed, it became clear that the significance of its changes merited a major release, and Laravel 5 was released in February 2015.

Laravel 5 featured a revamped directory structure, removal of the form and HTML helpers, the introduction of the contract interfaces, a spate of new views, Socialite for social media authentication, Elixir for asset compilation, Scheduler to simplify cron, dotenv for simplified environment management, form requests, and a brand new REPL (read–evaluate–print loop).

## What's So Special About Laravel?

So what is it that sets Laravel apart? Why is it worth having more than one PHP framework at any time? They all use components from Symfony anyway, right? Let's talk a bit about what makes Laravel "tick."

## The Philosophy of Laravel

You only need to read through the Laravel marketing materials and READMEs to start seeing its values. Taylor uses light-related words like "Illuminate" and "Spark."

And then there are these: “Artisans.” “Elegant.” Also, these: “Breath of fresh air.” “Fresh start.” And finally: “Rapid.” “Warp speed.”

The two most strongly communicated values of the framework are to increase developer speed and developer happiness. Taylor has described the “Artisan” language as intentionally contrasting against more utilitarian values. You can see the genesis of this sort of thinking in [his 2011 question on StackExchange](#) in which he stated, “Sometimes I spend ridiculous amounts of time (hours) agonizing over making code look pretty”—just for the sake of a better experience of looking at the code itself. And he’s often talked about the value of making it easier and quicker for developers to take their ideas to fruition, getting rid of unnecessary barriers to creating great products.

Laravel is, at its core, about equipping and enabling developers. Its goal is to provide clear, simple, and beautiful code and features that help developers quickly learn, start, and develop, and write code that’s simple, clear, and will last.

The concept of targeting developers is clear across Laravel materials. “Happy developers make the best code” is written in the documentation. “Developer happiness from download to deploy” was the unofficial slogan for a while. Of course, any tool or framework will say it wants developers to be happy. But having developer happiness as a *primary* concern, rather than secondary, has had a huge impact on Laravel’s style and decision-making progress. Where other frameworks may target architectural purity as their primary goal, or compatibility with the goals and values of enterprise development teams, Laravel’s primary focus is on serving the individual developer.

## How Laravel Achieves Developer Happiness

Just saying you want to make developers happy is one thing. Doing it is another, and it requires you to question what in a framework is most likely to make developers unhappy and what is most likely to make them happy. There are a few ways Laravel tries to make developers’ lives easier.

First, Laravel is a rapid application development framework. That means it focuses on a shallow (easy) learning curve and on minimizing the steps between starting a new app and publishing it. All of the most common tasks in building web applications, from database interactions to authentication to queues to email to caching, are made simpler by the components Laravel provides. But Laravel’s components aren’t just great on their own; they provide a consistent API and predictable structures across the entire framework. That means that, when you’re trying something new in Laravel, you’re more than likely going to end up saying, “... and it just works.”

This doesn’t end at the framework itself, either. Laravel provides an entire ecosystem of tools for building and launching applications. You have Homestead and Valet for local development, Forge for server management, and Envoyer for advanced deployment. And there’s a suite of add-on packages: Cashier for payments and subscrip-

tions, Echo for WebSockets, Scout for search, Passport for API authentication, Socialite for social login, and Spark to bootstrap your SaaS. Laravel is trying to take the repetitive work out of developers' jobs so they can do something unique.

Next, Laravel focuses on “convention over configuration”—meaning that if you’re willing to use Laravel’s defaults, you’ll have to do much less work than with other frameworks that require you to declare all of your settings even if you’re using the recommended configuration. Projects built on Laravel take less time than those built on most other PHP frameworks.

Laravel also focuses deeply on simplicity. It’s possible to use dependency injection and mocking and the Data Mapper pattern and repositories and Command Query Responsibility Segregation and all sorts of other more complex architectural patterns with Laravel, if you want. But while other frameworks might suggest using those tools and structures on every project, Laravel and its documentation and community lean toward starting with the simplest possible implementation—a global function here, a facade there, ActiveRecord over there. This allows developers to create the simplest possible application to solve for their needs.

An interesting source of how Laravel is different is that its creator and its community are more connected to and inspired by Ruby and Rails and functional programming languages than by Java. There’s a strong current in modern PHP to lean toward verbosity and complexity, embracing the more Java-esque aspects of PHP. But Laravel tends to be on the other side, embracing expressive, dynamic, and simple coding practices and language features.

## The Laravel Community

If this book is your first exposure to the Laravel community, you have something special to look forward to. One of the distinguishing elements of Laravel, which has contributed to its growth and success, is the welcoming, teaching community that surrounds it. From Jeffrey Way’s [Laracasts](#) video tutorials to [Laravel News](#) to Slack and IRC channels, from Twitter friends to bloggers to the Laracon conferences, Laravel has a rich and vibrant community full of folks who’ve been around since day one and folks who are on their own day one. And this isn’t an accident:

From the very beginning of Laravel, I’ve had this idea that all people want to feel like they are part of something. It’s a natural human instinct to want to belong and be accepted into a group of other like-minded people. So, by injecting personality into a web framework and being really active with the community, that type of feeling can grow in the community.

—Taylor Otwell, *Product and Support interview*

Taylor understood from the early days of Laravel that a successful open source project needed two things: good documentation and a welcoming community. And those two things are now hallmarks of Laravel.

## How It Works

Up until now, everything I've shared here has been entirely abstract. What about the code, you ask? Let's dig into a simple application ([Example 1-1](#)) so you can see what working with Laravel day-to-day is actually like.

*Example 1-1. “Hello, World” in routes/web.php*

```
// File: routes/web.php
<?php

Route::get('/', function() {
    return 'Hello, World!';
});
```

The simplest possible action you can take in a Laravel application is to define a route and return a result any time someone visits that route. If you initialize a brand new Laravel application on your machine, define the route in [Example 1-1](#), and then serve the site from the *public* directory, you'll have a fully functioning “Hello, World” example (see [Figure 1-1](#)).



*Figure 1-1. Returning “Hello, World!” with Laravel*

It looks very similar to do the same with controllers, as you can see in [Example 1-2](#).

*Example 1-2. “Hello, World” with controllers*

```
// File: routes/web.php
<?php

Route::get('/', 'WelcomeController@index');

// File: app/Http/Controllers/WelcomeController.php
<?php
namespace app\Http\Controllers;
```

```
class WelcomeController
{
    public function index()
    {
        return 'Hello, World!';
    }
}
```

And if we're storing our greetings in the database, it'll also look pretty similar (see [Example 1-3](#)).

*Example 1-3. Multigreeting "Hello, World" with database access*

```
// File: routes/web.php
<?php

Route::get('/', function() {
    return Greeting::first()->body;
});

// File: app/Greeting.php
<?php

use Illuminate\Database\Eloquent\Model;

class Greeting extends Model {}

// File: database/migrations/2015_07_19_010000_create_greetings_table.php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;

class CreateGreetingsTable extends Migration
{
    public function up()
    {
        Schema::create('greetings', function (Blueprint $table) {
            $table->increments('id');
            $table->string('body');
            $table->timestamps();
        });
    }

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

**Example 1-3** might be a bit overwhelming, and if so, just skip over it. We'll learn about everything that's happening here in later chapters, but you can already see that with just a few lines of code, we've set up database migrations and models and pulled records out. It's just that simple.

## Why Laravel?

So—why Laravel?

Because Laravel helps you bring your ideas to reality with no wasted code, using modern coding standards, surrounded by a vibrant community, with an empowering ecosystem of tools.

And because you, dear developer, deserve to be happy.

# Want to read more?

You can [buy this book](#) at oreilly.com  
in print and ebook format.

**Buy 2 books, get the 3rd FREE!**

Use discount code OPC10

All orders over \$29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer, including  
the iBookstore, the [Android Marketplace](#), and [Amazon.com](#).



O'REILLY®