O'REILLY®

Übersetzung der 2. US-Auflage

Laravel

Die umfassende Einführung

Das Framework für moderne PHP-Entwicklung

Matt Stauffer

Übersetzung von Jens Olaf Koch

ÜBERSETZUNG DER 2. AMERIKANISCHEN AUFLAGE

Laravel
Die umfassende Einführung

Das Framework für moderne PHP-Entwicklung

Matt Stauffer

Deutsche Übersetzung von
Jens Olaf Koch

O'REILLY®

Matt Stauffer

Lektorat: Ariane Hesse
Übersetzung: Jens Olaf Koch
Fachlicher Review: Mitarbeiter von mindtwo, *www.mindtwo.de*
Korrektorat: Claudia Lötschert, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Michael Oreal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-129-5
PDF 978-3-96010-374-5
ePub 978-3-96010-372-1
mobi 978-3-96010-373-8

1. Auflage 2020

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of titled *Laravel: Up & Running*, 2E
ISBN 9781492041214 © 2019 Matt Stauffer.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: kommentar@oreilly.de.

5 4 3 2 1 0

Dieses Buch ist meiner Familie gewidmet.
Mia, meiner kleinen Prinzessin, diesem Bündel voller Freude und Energie.
Malachi, meinem kleinen Prinzen, Abenteurer und Empathen.
Tereva, meiner Inspiration, Ermutigerin, Upgraderin, Motivatorin, Rippe.

Inhalt

Vorwort

Der Weg, der mich zu Laravel geführt hat, ist ganz typisch: Ich hatte jahrelang in PHP programmiert, war aber bereits dabei, mich davon zu verabschieden und mich den Möglichkeiten von Rails und anderen modernen Web-Frameworks zu widmen. Insbesondere Rails besaß eine lebendige Community und bot eine perfekte Kombination aus meinungsstarken Vorgaben und Flexibilität sowie das Potenzial von Ruby Gems, vorgefertige Packages mit Programmen und Bibliotheken zu nutzen.

Aber etwas hielt mich noch davon ab, PHP endgültig hinter mir zu lassen, und ich war froh darüber, als ich auf Laravel stieß. Laravel bot alles, was mich an Rails anzog, aber es war mehr als bloß ein Rails-Klon. Es war ein innovatives Framework mit unglaublich guter Dokumentation, einer einladenden Community und deutlichen Einflüssen verschiedener Sprachen und Frameworks.

Seitdem habe ich in Blogs, Podcasts und Vorträgen auf Konferenzen geteilt, was ich auf meiner Reise mit Laravel gelernt habe; ich habe Dutzende von Laravel-Apps für Arbeits- und Nebenprojekte geschrieben; und ich habe Tausende von Laravel-Entwicklern online und persönlich getroffen. Es gibt viele Tools in meinem Entwicklungs-Werkzeugkasten, aber ich bin ehrlich gesagt am glücklichsten, wenn ich mich vor eine Kommandozeile setzen und `laravel new` *`projektName`* eingeben kann.

Worum es in diesem Buch geht

Dies ist nicht das erste Buch über Laravel, und es wird nicht das letzte sein. Es soll kein Buch sein, das jede Zeile Code oder jedes mögliche Implementierungsmuster abdeckt. Und es soll kein Buch sein, das automatisch veraltet ist, sobald eine neue Version von Laravel veröffentlicht wird. Stattdessen geht es in erster Linie darum, Entwicklern einen wirklichen Überblick und konkrete Beispiele zu geben, damit sie wissen, was sie benötigen, um in allen Laravel-Releases mit jedem einzelnen Feature und Subsystem arbeiten zu können. Anstatt einfach die Dokumentation nachzuerzählen, möchte ich Ihnen helfen, die grundlegenden Konzepte zu verstehen, auf denen Laravel aufbaut.

Laravel ist ein leistungsfähiges und flexibles PHP-Framework. Es gibt eine florierende Community und ein umfassendes Ökosystem an Werkzeugen, was dazu beiträgt, dass Laravels Attraktivität und Reichweite ständig wachsen. Dieses Buch richtet sich an Entwickler, die bereits wissen, wie man Websites und Anwendungen erstellt, und die jetzt lernen wollen, wie man genau das erfolgreich auch mit Laravel macht.

Die Laravel-Dokumentation ist umfassend und ausgezeichnet. Wenn Sie feststellen, dass ich ein bestimmtes Thema Ihrem Geschmack zufolge nicht gründlich genug abdecke, empfehle ich Ihnen, die Onlinedokumentation (*https://laravel.com/docs*) zu besuchen und tiefer in dieses spezielle Thema einzutauchen.

In diesem Buch erwartet Sie eine angenehme Balance zwischen einer Einführung auf hohem Niveau und konkreter Anwendung, und am Ende werden Sie sich hoffentlich dabei wohlfühlen, wenn Sie eine komplette Anwendung in Laravel von Grund auf neu schreiben. Und wenn ich meinen Job gut gemacht habe, werden Sie auch heiß darauf sein.

Für wen dieses Buch gedacht ist

Dieses Buch setzt Kenntnisse grundlegender objektorientierter Programmierpraktiken, PHP (oder zumindest der allgemeinen Syntax der Sprachen der C-Familie), grundlegender Konzepte des Entwurfsmusters »Model-View-Controller« (MVC, auf Deutsch »Modell-Präsentation-Steuerung«) und der Arbeit mit Template Engines voraus. Wenn Sie zuvor noch nie selbst eine Website entworfen haben, wird es vielleicht etwas zu anspruchsvoll sein. Aber solange Sie über Programmiererfahrung verfügen, müssen Sie vor der Lektüre dieses Buchs kein Vorwissen zu Laravel haben – wir decken alles ab, was Sie wissen müssen, angefangen beim einfachsten »Hallo, Welt!«.

Laravel kann mit jedem Betriebssystem eingesetzt werden, aber es wird einige Kommandozeilen-Befehle im Buch geben, die am einfachsten unter Linux/macOS ausgeführt werden können. Windows-Benutzer werden es mit diesen Befehlen und mit moderner PHP-Entwicklung möglicherweise etwas schwerer haben, aber wenn Sie Homestead (eine virtuelle Linux-Maschine) zum Laufen bringen, können Sie alle Befehle von dort aus ausführen.

Wie dieses Buch aufgebaut ist

Dieses Buch ist in gewisser Weise chronologisch gegliedert: Wenn Sie Ihre erste Webanwendung mit Laravel erstellen, decken die ersten Kapitel die grundlegenden Komponenten ab, die Sie zum Einstieg benötigen, während die späteren Kapitel weniger grundlegende bzw. etwas ausgefallenere Funktionen behandeln.

Jeder Abschnitt des Buchs kann für sich allein gelesen werden, aber für alle, die dieses PHP-Framework zum ersten Mal einsetzen, habe ich versucht, die Kapitel so zu strukturieren, dass es sinnvoll ist, alles in der vorgegebenen Reihenfolge zu lesen.

Wenn es passt, schließt ein Kapitel mit zwei speziellen Abschnitten: »Testen« und »TL;DR.« Falls Sie es nicht kennen: »TL;DR« steht für »too long; didn't read«, also »zu lang, nicht gelesen«. Diese letzten Abschnitte zeigen, wie Sie Tests für die im Kapitel behandelten Funktionen schreiben können, und fassen sehr kompakt zusammen, um was es im betreffenden Kapitel geht.

Dieses Buch ist für Laravel 6 geschrieben, deckt aber in der Regel Funktionen und Syntaxänderungen von Laravel 5.1 an ab – auch um die permanente Weiterentwicklung des Frameworks lebendig darzustellen.

Über die zweite Ausgabe

Die erste Ausgabe von *Laravel: Up & Running* erschien im November 2016 und umfasste die Laravel-Versionen 5.1 bis 5.3. Diese zweite Ausgabe deckt die Versionen 5.4 bis 6.6 sowie Laravel Dusk und Horizon ab, zudem ist ein 18. Kapitel über Community-Ressourcen und andere Laravel-Pakete hinzugekommen, die nicht zum Kern des Frameworks gehören und in den ersten 17 Kapiteln nicht behandelt wurden.

Vorbemerkung zur deutschen Ausgabe

Laravel gibt es nur in einer englischsprachigen Version, und das gilt auch für nahezu alle zusätzlichen Module, Programmpakete und Dienstleistungsangebote des gesamten Laravel-Ökosystems.

Wir haben deshalb bei der Übersetzung dieses Buchs versucht, eine angenehme Balance zwischen englischen Originalbegriffen und deutschen Fachausdrücken zu erreichen. Zumal es für viele der in Programmierung und Anwendungsentwicklung benutzten Begrifflichkeiten gar keine deutsche Entsprechung gibt. Deshalb benutzen wir mal die englischen, mal die deutschen Begriffe, mit einer Tendenz zum Englischen. Da Sie sich als angehender Laravel-Entwickler überwiegend im englischsprachigen Umfeld bewegen werden, wäre es eher kontraproduktiv, würden wir zu viele Begriffe oder Teile von Programmlistings eindeutschen.

Dazu kommt, dass Laravel von sich aus einige Funktionen mitbringt, die selbstständig Pluralformen (zum Beispiel von Modell-Namen) erstellen. Dabei wird standardmäßig ein »s« benutzt und angehängt. Obwohl man dieses Verhalten individuell übersteuern kann, liest sich der Code insgesamt geschmeidiger, und man hat weniger Arbeit, wenn man Laravel seinen »Willen« lässt.

Der Schöpfer von Laravel, Taylor Otwell, gibt sich extrem viel Mühe, den Code so lesbar und so nah an natürlicher Sprache zu halten, wie es nur geht. Und es liest

sich natürlich besser, wenn eine Datenbank-Migration *create_customers_table* heißt und nicht *create_kundens_table* (ohne Übersteuerung des Standardverhaltens) oder *create_kunden_table* (mit angepasster Pluralform).

Konventionen, die in diesem Buch verwendet werden

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Kursiv
: Zeigt neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen an.

`Nichtproportional`
: Wird für Programm-Listings verwendet, aber auch innerhalb von Absätzen, um dort auf Programmelemente wie Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter zu verweisen.

`Nichtproportional fett`
: Zeigt Befehle oder anderen Text an, der vom Benutzer wortgetreu eingegeben werden muss.

`Nichtproportional kursiv`
: Zeigt Programmcode an, der durch Benutzereingaben oder durch kontextabhängige Werte ersetzt werden soll.

{Kursiv in Klammern}
: Zeigt Dateinamen oder Dateipfade an, die durch Benutzereingaben oder durch kontextabhängige Werte ersetzt werden sollen.

Dieses Element weist auf einen Tipp oder Vorschlag hin.

Dieses Element kennzeichnet einen allgemeinen Hinweis.

Dieses Element weist auf eine Warnung hin.

O'Reilly Online-Lernen

Seit fast 40 Jahren bietet *O'Reilly Media* Technologie- und Business-Training, Wissen und Einsichten, um Unternehmen zum Erfolg zu verhelfen.

Unser einzigartiges Netzwerk von Experten und Innovatoren teilt sein Wissen und seine Expertise in Büchern, Artikeln, Konferenzen und auf unserer Online-Lernplattform. Die Online-Lernplattform von O'Reilly bietet Ihnen On-Demand-Zugriff auf Live-Schulungen, detaillierte Lernpfade, interaktive Codierumgebungen und eine umfangreiche Sammlung von Texten und Videos von O'Reilly und über 200 anderen Verlagen. Für weitere Informationen besuchen Sie bitte *https://www.oreilly.com*.

Englischsprachige Website zu diesem Buch

Es gibt eine englischsprachige Website zu diesem Buch, auf der wir Errata, Beispiele und alle weiteren Informationen aufführen. Sie können diese Seite unter *https://bit.ly/laravel-up-and-running-2e* aufrufen.

Danksagung für die erste Ausgabe

Dieses Buch wäre ohne die verständnisvolle Unterstützung meiner wunderbaren Frau Tereva und meines Sohnes Malachi (»Papa schreibt, Buddy!«) nicht möglich gewesen. Und obwohl sie es selbst nicht wirklich bewusst miterlebt hat, war meine Tochter Mia fast über die gesamte Zeit der Entstehung des Buchs dabei, sodass dieses Buch der ganzen Familie gewidmet ist. Es gab viele lange Abendstunden und Arbeitsbesuche bei Starbucks am Wochenende, sodass ich oft nicht bei meiner Familie war, und ich könnte nicht dankbarer sein für ihre Unterstützung und vor allem für ihr Da-Sein, das mein Leben einfach bereichert.

Darüber hinaus hat mich die gesamte Tighten-Familie während des Schreibens des Buchs unterstützt und ermutigt, und mehrere meiner Kollegen haben Codebeispiele bearbeitet (Keith Damiani, »Editor Extraordinaire«) und mir bei schwierigen Projekten geholfen (Adam Wathan, König der Collection-Pipeline). Dan Sheetz, mein Partner bei Tighten, war so freundlich, mir viele Stunden nachzusehen, die ich mit der Arbeit an diesem Buch verbracht habe, und war dabei immer eine Stütze und Ermutigung; und Dave Hicking, unser Betriebsleiter, half mir, meine Zeiteinteilung und Verantwortlichkeiten rund um meine Schreibzeiten zu organisieren.

Taylor Otwell verdient Dank und Anerkennung als Schöpfer von Laravel – und damit auch für die Schaffung vieler Jobs und dafür, dass so viele Entwickler ihr Leben noch mehr lieben. Er verdient Anerkennung dafür, wie sehr er sich auf die Zufriedenheit der Entwickler konzentriert und wie hart er gearbeitet hat, um mit

Empathie für uns Entwickler eine positive und ermutigende Community aufzubauen. Aber ich möchte ihm auch dafür danken, dass er ein liebenswürdiger, unterstützender und anregender Freund ist. Taylor, du bist ein Held.

Vielen Dank an Jeffrey Way, der einer der besten Lehrer im Internet ist. Er hat mich mit Laravel bekannt gemacht und bringt Laravel heute noch jeden Tag vielen Menschen näher. Er ist auch, wenig überraschend, ein fantastischer Mensch, den ich gerne Freund nenne.

Vielen Dank an Jess D'Amico, Shawn McCool, Ian Landsman und Taylor, die mich schon früh als Konferenzsprecher geschätzt haben und mir eine Plattform bieten, von der aus ich wirken kann. Vielen Dank an Dayle Rees, der in den frühen Tagen von Laravel so vielen Menschen dabei geholfen hat, es zu erlernen.

Vielen Dank an alle Menschen, die ihre Zeit und Mühe dem Schreiben von Blog-Posts über Laravel gewidmet haben, besonders in den Kindertagen von Laravel: Eric Barnes, Chris Fidao, Matt Machuga, Jason Lewis, Ryan Tablada, Dries Vints, Maks Surguy und viele mehr.

Und vielen Dank an die gesamte Community von Freunden auf Twitter, IRC und Slack, die über die Jahre mit mir kommuniziert haben. Ich wünschte, ich könnte jeden Namen nennen, aber ich würde einige vergessen und mich dann dafür schämen. Ihr seid alle brillant, und ich fühle mich geehrt, regelmäßig mit euch in Kontakt zu stehen.

Vielen Dank an meine O'Reilly-Lektorin, Ally MacDonald, und alle meine technischen Redakteure: Keith Damiani, Michael Dyrynda, Adam Fairholm und Myles Hyson.

Und natürlich danke an den Rest meiner Familie und meine Freunde, die mich direkt oder indirekt unterstützt haben – meine Eltern und Geschwister, die Gainesville-Community, Geschäftspartner und Autoren, andere Konferenzredner und die einzigartige DCB. Ich muss aufhören, weiter zu schreiben, bevor ich noch meinen Starbucks-Baristas danke.

Danksagung für die zweite Ausgabe

Die zweite Ausgabe ist der ersten sehr ähnlich, sodass alle vorherigen Danksagungen noch gültig sind. Aber ich habe diesmal Hilfe von ein paar weiteren Personen bekommen. Meine technischen Korrekturleser waren Tate Peñaranda, Andy Swick, Mohamed Said und Samantha Geitz, und meine neue O'Reilly-Lektorin war Alicia Young, die mich im letzten Jahr über viele Veränderungen in meinem Leben und innerhalb der Laravel-Community hinweg bei der Arbeit gehalten hat. Matt Hacker vom Atlas-Team beantwortete alle meine dummen AsciiDoc-Formatierungsfragen, auch zur überraschend schwierigen Formatierung der `__()`-Methode.

Und ohne die Hilfe meines Assistenten Wilbur Powery hätte ich es nicht geschafft, diese zweite Ausgabe zu schreiben. Wilbur war bereit, die Changelogs,

Pull-Requests und Ankündigungen der letzten Jahre zu durchsuchen und jedes Feature der aktuellen Struktur des Buchs zuzuordnen, und testete tatsächlich jedes einzelne Codebeispiel, damit ich meine begrenzte Zeit und Energie auf das Schreiben der neuen und aktualisierten Abschnitte konzentrieren konnte.

Außerdem hat meine Tochter Mia jetzt den Weg aus dem Bauch ihrer Mutter gefunden. Also lassen Sie mich einfach noch ihre Freude und Energie und Liebe und Niedlichkeit und ihren Abenteuergeist der Liste meiner Inspirationsquellen hinzufügen.

KAPITEL 1

Warum Laravel?

In den frühen Tagen des dynamischen World Wide Web sah das Schreiben einer Anwendung ganz anders aus als heute. Die Entwickler waren damals nicht nur dafür verantwortlich, den Code für die jeweils spezielle Anwendungslogik zu schreiben, sondern auch für all jene Komponenten, die immer wieder für Websites benötigt werden: für die Authentifizierung von Benutzern, die Validierung von Eingaben, Datenbankzugriffe, die Erstellung von Vorlagen und vieles mehr.

Heutzutage gibt es Dutzende von Frameworks für die Anwendungsentwicklung und Tausende von Komponenten und Bibliotheken, die allen Programmierern leicht zugänglich sind. Es ist gängige Rede unter Entwicklern, dass, kaum habe man ein Framework gelernt, es bereits drei neuere (und angeblich bessere) Frameworks gebe, die es gerne ersetzen möchten.

»Weil er halt existiert«, mag eine sinnvolle Rechtfertigung dafür sein, einen Berg zu besteigen. Aber es gibt bessere Gründe als »Weil es halt existiert«, um sich für ein bestimmtes Framework zu entscheiden – oder überhaupt eines zu verwenden. Stellen wir uns also die berechtigte Frage: Wozu sind Frameworks eigentlich gut? Genauer gesagt: Wozu ist Laravel gut?

Warum ein Framework verwenden?

Es ist leicht nachzuvollziehen, warum es hilfreich ist, die einzelnen Komponenten und Pakete zu verwenden, die es für PHP-Entwickler gibt. Bei Packages ist jemand anderes für die Entwicklung und Wartung eines isolierten Stücks Programmcode verantwortlich, mit dem eine bestimmte Aufgabe gelöst wird, und theoretisch sollte diese Person ein tieferes Verständnis für diese einzelne Komponente besitzen, als Sie sich selbst in angemessener Zeit aneignen können.

Frameworks wie Laravel – und Symfony, Lumen und Slim – versammeln eine Reihe Komponenten von Drittanbietern und bündeln diese mit Framework-eigenem »Klebstoff« wie Konfigurationsdateien, Service Providern, vordefinierten Verzeichnisstrukturen und Anwendungs-Bootstraps. Ganz allgemein besteht der Vorteil, ein Framework zu verwenden, darin, dass jemand für Sie nicht nur bereits über einzelne

Komponenten entschieden hat, sondern auch darüber, wie diese Komponenten zusammenarbeiten sollen.

»Ich baue es einfach selbst«

Nehmen wir an, Sie beginnen mit einer neuen Webapplikation, ohne die Vorteile eines Frameworks zu nutzen. Womit fangen Sie an? Ziemlich sicher müssen HTTP-Requests geroutet werden, sodass Sie sich alle verfügbaren HTTP-Request- und Response-Bibliotheken anschauen und danach eine auswählen müssen. Dann brauchen Sie einen Router. Und wahrscheinlich benötigen Sie auch irgendeine Art Konfigurationsdatei für die Routen. Welche Syntax soll dabei verwendet werden? An welcher Stelle soll die Konfigurationsdatei abgelegt werden? Und was ist mit Controllern? Wo sollen die liegen und wie sollen sie geladen werden? Wahrscheinlich benötigen Sie einen Dependency Injection Container, um die Controller und ihre Abhängigkeiten aufzulösen. Aber welchen?

Wenn Sie sich die Zeit nehmen, all diese Fragen zu beantworten und Ihre Anwendung erfolgreich zu erstellen, was bedeutet das dann für einen möglichen späteren Entwickler? Und was machen Sie, wenn Sie beispielsweise vier dieser Anwendungen mit solch einem Framework Marke Eigenbau programmiert haben – oder gar ein ganzes Dutzend – und Sie sich jeweils daran erinnern müssen, wo sich die Controller befinden oder wie genau die Routing-Syntax lautet?

Konsistenz und Flexibilität

Frameworks lösen dieses Problem, indem sie eine sorgfältig durchdachte Antwort auf die Frage »Welche Komponente sollen wir verwenden?« geben und zudem sicherstellen, dass die jeweils gewählten Komponenten auch gut zusammenarbeiten. Darüber hinaus bieten Frameworks feste Konventionen, die die Menge an Code reduzieren, die ein Entwickler, der neu zu einem Projekt stößt, verstehen muss: Sobald Sie begriffen haben, wie Routing in *einem* Laravel-Projekt funktioniert, wissen Sie sofort, wie es in *allen* Laravel-Projekten funktioniert.

Wenn jemand vorschreibt, dass sein eigenes Framework für jedes neue Projekt benutzt werden muss, dann geht es letztlich darum, zu *kontrollieren*, was in die Grundlage seiner Applikation einfließt und was nicht. Das bedeutet, dass Ihnen die besten Frameworks nicht nur ein solides Fundament bieten, sondern auch die Freiheit geben, praktisch alles Ihren eigenen Wünschen entsprechend anzupassen. Und genau das ist es, was Laravel so besonders macht und was ich Ihnen im weiteren Verlauf dieses Buchs zeigen möchte.

Eine kurze Geschichte der Web- und PHP-Frameworks

Will man die Frage »Warum Laravel?« beantworten, muss man seine Geschichte kennen – und seine Vorläufer. Bevor Laravel populär wurde, gab es bereits eine

Vielzahl von Frameworks und anderer Entwicklungen in PHP und verwandten Bereichen.

Ruby on Rails

David Heinemeier Hansson veröffentlichte 2004 die erste Version von Ruby on Rails, und seitdem ist fast jedes Web Application Framework in irgendeiner Weise von Rails beeinflusst.

Rails popularisierte MVC, das Model-View-Controler-Paradigma, aber auch RESTful JSON APIs, die Regel »Konvention vor Konfiguration«, ActiveRecord und viele weitere Tools und Konventionen, die einen großen Einfluss auf die Art und Weise hatten, wie Webentwickler an ihre Anwendungen herangingen – insbesondere im Hinblick auf die schnelle Anwendungsentwicklung.

Eine Welle von PHP-Frameworks

Es war den meisten Entwicklern klar, dass Rails und ähnliche Frameworks die Zukunft darstellten, und schnell erschienen PHP-Frameworks auf der Bildfläche, einschließlich derjenigen, die zugegebenermaßen Rails imitierten.

CakePHP war im Jahr 2005 das erste, und es folgten bald Symfony, CodeIgniter, Zend Framework und Kohana (ein CodeIgniter-Fork). 2008 kam Yii, und 2010 erschienen Aura und Slim. 2011 sah dann die Geburt von FuelPHP und Laravel, die beide als CodeIgniter-Alternativen vorgeschlagen wurden.

Einige dieser Frameworks orientierten sich eher an Rails und konzentrierten sich auf objektrelationale Abbildung (Object-relational mapper, ORM), MVC-Strukturen und andere Tools für eine rapide Entwicklung. Andere – wie Symfony und Zend – konzentrierten sich mehr auf Enterprise Design Patterns und E-Commerce.

Das Gute und das Schlechte an CodeIgniter

CakePHP und CodeIgniter waren die beiden frühen PHP-Frameworks, bei denen klar kommuniziert wurde, wie sehr sie von Ruby on Rails inspiriert waren. CodeIgniter wurde schnell bekannt und war 2010 wohl das beliebteste der unabhängigen PHP-Frameworks.

CodeIgniter war simpel, einfach zu bedienen, besaß eine erstaunlich gute Dokumentation und wurde von einer starken Community gestützt. Aber der Einsatz moderner Technologien und Entwurfsmuster schritt nur langsam voran. Als die Framework-Welt wuchs und die PHP-Tools weiterentwickelt wurden, begann CodeIgniter sowohl in Bezug auf den technologischen Fortschritt als auch auf die Funktionen, die es out of the box mitbrachte, ins Hintertreffen zu geraten. Im Gegensatz zu vielen anderen Frameworks wurde CodeIgniter von einem Unternehmen verwaltet, und es dauerte einige Zeit, bis man die neueren Features von PHP 5.3 wie Namespaces, die Nutzung von GitHub und später Composer aufgriff. Im

Jahr 2010 war Taylor Otwell, Laravels Schöpfer, schließlich so unzufrieden mit CodeIgniter, dass er begann, sein eigenes Framework zu schreiben.

Laravel 1, 2 und 3

Die erste Beta von Laravel 1 wurde im Juni 2011 veröffentlicht und war eine komplette Neuentwicklung. Sie beinhaltete einen eigenen objektrelationalen Mapper namens Eloquent, ein Closure-basiertes Routing (inspiriert von Ruby Sinatra), ein modulares Erweiterungssystem und Hilfsfunktionen für Formulare, Validierung, Authentifizierung und vieles mehr.

Die frühe Entwicklung von Laravel ging schnell voran, und Laravel 2 und 3 wurden bereits im November 2011 bzw. Februar 2012 veröffentlicht. In diesen Versionen wurden Controller, Modul-Tests, ein Befehlszeilen-Tool, ein Container mit Inversion of Control (Steuerungsumkehr), Relationen zwischen Eloquent-Modellen sowie Migrationen eingeführt.

Laravel 4

Für Laravel 4 hat Taylor Otwell das gesamte Framework noch einmal von Grund auf neu geschrieben. Zu diesem Zeitpunkt war Composer, der inzwischen allgegenwärtige Paketmanager von PHP, bereits auf dem Weg dazu, sich zu einem Industriestandard zu entwickeln, und Taylor sah einen großen Mehrwert darin, das Framework als eine Sammlung von Komponenten zu konzipieren, die von Composer verteilt und zu einem Ganzen gebündelt wurden.

Er entwickelte eine Reihe von Komponenten unter dem Codenamen *Illuminate* und brachte im Mai 2013 Laravel 4 heraus, das eine völlig neue Struktur besaß. Anstatt das Framework als einen großen Download anzubieten, zieht sich Laravel nun die meisten seiner Komponenten per Composer aus Symfony, einem anderen PHP-Framework, das seine Module zur Verwendung durch andere freigegeben hat, und kombiniert diese mit den eigenen Illuminate-Komponenten.

Laravel 4 führte auch erstmals Warteschlangen, eine Mail-Komponente, Fassaden und Datenbank-Seedings ein. Und da Laravel nun auf Symfony-Komponenten setzte, wurde angekündigt, dass Laravel den halbjährlichen Release-Terminplan von Symfony widerspiegeln würde (nicht exakt, aber zeitnah).

Laravel 5

Laravel 4.3 sollte im November 2014 veröffentlicht werden, aber im Laufe der Entwicklung wurde deutlich, dass den Änderungen eine größere Bedeutung zukam und damit ein Versionssprung nahelag, und so wurde es im Februar 2015 als Laravel 5 herausgebracht.

Für Laravel 5 wurden die Verzeichnisstruktur überarbeitet, die Formular- und HTML-Helper entfernt sowie Contract Interfaces, eine Flut neuer Views, Socialite

für eine Social-Media-Authentifizierung, Elixir für die Zusammenstellung von Assets, Scheduler zur Vereinfachung von Cron, Dotenv für vereinfachtes Umgebungsmanagement, Verbesserungen bei der Validierung von Formulareingaben und eine brandneue REPL (Read-Evaluate-Print-Loop) eingeführt. Seitdem ist die Funktionalität von Laravel weiter ausgereift, aber es gab keine größeren Änderungen mehr wie in früheren Versionen.

Laravel 6

Laravel 6 erschien im September 2019. Die sichtbarste Veränderung bestand darin, dass bei der Versionsbenennung auf Semantic Versioning (*https://semver.org/lang/de/*) umgestellt wurde.

Vor Laravel 6 lag zwischen zwei sogenannten Major Releases ungefähr eine Zeitspanne von sechs Monaten, beispielsweise zwischen den Versionen 5.4 und 5.5. Die neue semantische Versionierung bedeutet, dass jedes Mal, wenn ein neues rückwärtskompatibles Feature zum Framework hinzugefügt wird, die zweite Ziffer der Versionsnummer erhöht wird. Deshalb folgen seit Version 6.0 in schneller Folge 6.1, 6.2 usw. Das ist allerdings nur eine optische Beschleunigung, die inhaltliche Fortentwicklung von Laravel geht weiterhin im gleichen Rhythmus vonstatten.

Neben der semantischen Versionierung brachte Version 6 unter anderem eine deutlich verbesserte Fehlerseite namens Ignition, die neue Job-Middleware, Lazy Collections, Verbesserungen bei Eloquent-Subqueries und die Separierung des Frontend-Scaffoldings in ein separates Paket.

Zudem wurde das Laravel-Ökosystem durch Laravel Vapor bereichert, eine serverlose Deployment-Plattform, die als Infrastruktur Amazons AWS Lambda nutzt.

Was ist so besonders an Laravel?

Was genau ist es letztlich, was Laravel so außergewöhnlich macht? Wozu soll es gut sein, dass es mehr als ein PHP-Framework gibt? Alle Frameworks verwenden doch sowieso Komponenten von Symfony, oder nicht? Lassen Sie uns also ein wenig darüber reden, was Laravel ausmacht.

Die Philosophie von Laravel

Schon wenn man das Marketingmaterial zu Laravel und die Readme-Dateien liest, bekommt man einen Eindruck von den »inneren Werten« des Frameworks. Taylor verwendet Wörter wie »Illuminate« (dt. *Erleuchtung*) und »Spark« (dt. *Funke*), die sich um das Thema »Licht« drehen. Und dann gibt es noch diese Begriffe: »Artisan« »Elegant« Und diese: »Frische Luft zum Atmen« »Neuanfang« Und schließlich: »Schnell« »Warp-Geschwindigkeit«.

Die beiden am stärksten kommunizierten Werte des Frameworks liegen in der beabsichtigten Steigerung der Entwicklungsgeschwindigkeit und der Entwickler-

zufriedenheit. Die Sprache »Artisan« (dt. *Handwerker*) beschreibt Taylor Otwell als bewusst in Kontrast stehend zu eher funktionalen Werten. Die Anfänge dieses Denkens kann man in einem seiner Postings auf StackExchange (*https://bit.ly/2dT5kmS*) sehen, in dem er sagte: »Manchmal verbringe ich geradezu lächerlich viel und quälende Zeit damit, Code ›hübsch aussehen zu lassen‹« – nur damit es sich besser anfühlt, wenn man den Code als solchen betrachtet. Und er hat oft über den Wert gesprochen, der darin liegt, es Entwicklern leichter zu machen, ihre Ideen zu verwirklichen, und unnötige Hindernisse für die Entwicklung großartiger Produkte zu beseitigen.

Bei Laravel geht es im Kern darum, Entwickler für ihre Arbeit besser auszustatten und zu befähigen. Taylors Ziel ist es, klaren, einfachen und schönen Code bereitzustellen sowie Features, die Entwicklern helfen, schnell zu lernen, loszulegen und zu entwickeln und Code zu produzieren, der einfach, klar und dauerhaft nutzbar ist.

Das Konzept, die Entwickler in den Fokus zu stellen, wird in allen Materialien deutlich, die es zu Laravel gibt. »Zufriedene Entwickler schreiben den besten Code« steht in der Dokumentation. »Entwicklerzufriedenheit vom Download bis zum Deployment« (»Developer happiness from download to deploy«) war eine Zeit lang der inoffizielle Slogan. Natürlich wird jedes Tool oder Framework mit dem Anspruch vermarktet, dass die Zufriedenheit von Entwicklern im Mittelpunkt steht. Aber dass die Zufriedenheit der Entwickler als *primäres* und nicht nur als sekundäres Anliegen derart proklamiert wurde, hatte einen großen Einfluss auf den gesamten Stil von Laravel und die Art und Weise, wie Entscheidungen getroffen wurden. Während andere Frameworks in erster Linie die architektonische Reinheit oder die Kompatibilität mit den Wünschen und Anforderungen von Entwicklungsteams größerer Unternehmen anstreben, liegt das Hauptaugenmerk von Laravel auf der Unterstützung des einzelnen Entwicklers. Das bedeutet nicht, dass Sie in Laravel keine architektonisch reinen oder unternehmenstauglichen Anwendungen schreiben können, aber das muss nicht auf Kosten der Lesbarkeit und Verständlichkeit Ihrer Codebasis gehen.

Wie sich Laravel um die Zufriedenheit des Entwicklers verdient macht

Zu sagen, dass man Entwickler glücklich und zufrieden machen will, ist eine Sache. Es tatsächlich auch zu tun, ist etwas anderes. Und man muss sich dazu fragen, was in einem Framework Entwickler am ehesten unzufrieden und was sie zufrieden macht. Es gibt einige Ansätze, das Leben von Entwicklern zu erleichtern.

Erstens ist Laravel ein Framework für eine rapide Anwendungsentwicklung. Die Lernkurve ist relativ flach, und es wird versucht, mit möglichst wenigen Schritten von der ersten Einrichtung einer Anwendung zu ihrer Veröffentlichung zu kommen. Alle typischen Aufgaben beim Erstellen von Webanwendungen, von den

Interaktionen mit Datenbanken über Authentifizierung, Warteschlangen, E-Mails bis hin zum Caching, werden durch die von Laravel bereitgestellten Komponenten vereinfacht. Aber die Komponenten von Laravel sind nicht nur für sich jeweils allein betrachtet exzellent, sie bieten auch eine konsistente API und vorhersehbare Strukturen im gesamten Framework. Wenn Sie in Laravel etwas Neues ausprobieren, werden Sie am Ende wahrscheinlich sagen können: »... und es funktioniert einfach.«

Dieser Ansatz geht zudem über das Framework selbst hinaus. Um Laravel herum existiert ein ganzes Ökosystem von Tools für die Erstellung und Veröffentlichung von Anwendungen. Da gibt es Homestead und Valet für die lokale Entwicklung, Forge für die Serververwaltung und Envoyer für eine fortgeschrittene Softwareverteilung. Zudem gibt es eine Reihe von Add-on-Paketen: Cashier für Zahlungen und Abonnements, Echo für WebSockets, Scout für die Suche, Passport für API-Authentifizierung, Dusk für Frontend-Tests, Socialite für Social Login, Horizon für die Überwachung von Warteschlangen, Nova für den Aufbau von Admin-Panels und Spark für Software-as-a-Service-Angebote. Laravel versucht, uns Entwicklern repetitive Arbeit zu ersparen, damit wir mit der eingesparten Zeit etwas Sinnvolleres anfangen können.

Außerdem gilt in Laravel die Regel »Konvention vor Konfiguration«. Das bedeutet, dass es in Laravel viel einfacher als in anderen Frameworks ist, die Standardwerte zu verwenden, weil man dort oft sogar die empfohlenen Einstellungen explizit deklarieren muss. Projekte, die mit Laravel realisiert werden, sind damit schneller umzusetzen als mit den meisten anderen PHP-Frameworks.

Laravel legt zudem großen Wert auf Einfachheit. Natürlich kann man Dependency Injection und Mocking und Data-Mapping und Repositories und Command Query Responsibility Segregation und alle Arten von anderen komplexeren Architekturmustern mit Laravel verwenden, wenn man will. Aber während andere Frameworks die Verwendung dieser Tools und Strukturen für jedes Projekt vorschlagen, tendieren Laravel und seine Dokumentation und Community dazu, mit der einfachsten möglichen Implementierung zu beginnen – einer globalen Funktion hier, einer Fassade dort, ActiveRecord bei Bedarf. Damit können Entwickler mit der für die gewünschte Lösung einfachsten Anwendung beginnen, ohne in komplexen Umgebungen bei der späteren Erweiterung eingeschränkt zu sein.

Laravel unterscheidet sich von anderen PHP-Frameworks auch dadurch, dass sein Schöpfer und die Community als Ganze eher mit Ruby und Rails und funktionalen Programmiersprachen verbunden und von diesen inspiriert sind als von Java. Im modernen PHP gibt es eine starke Strömung, die eher der Ausführlichkeit und Komplexität zuneigt und damit die eher Java-ähnlichen Aspekte von PHP betont und ausbauen möchte. Laravel steht dagegen eher auf der anderen Seite und stellt ausdrucksstarke, dynamische und einfache Codierungspraktiken und Sprachmerkmale in den Vordergrund.

Die Laravel-Community

Wenn dieses Buch Ihre erste Begegnung mit der Laravel-Community sein sollte, dann freuen Sie sich auf etwas Besonderes. Sie macht einen wesentlichen Unterschied und ist eines der Elemente, die zum Wachstum und Erfolg von Laravel beigetragen haben: die einladende, unterstützende Gemeinschaft all derer, die das Framework nutzen. Von Jeffrey Ways Video-Tutorials auf Laracasts (*https://laracasts.com/*) über die Laravel News (*https://laravel-news.com/*) zu Slack und IRC und Discord-Kanälen, von Twitter-Freunden über Blogger zu Podcasts und den Laracon-Konferenzen – Laravel besitzt eine große und lebendige Gemeinschaft voller Menschen, die oft seit dem ersten Tag dabei sind, und solchen, die gerade erst ihren eigenen »Tag 1« mit Laravel erleben. Und das ist kein Zufall:

> Von Anfang an hatte ich die Vorstellung, dass alle Menschen gerne das Gefühl haben, Teil von etwas zu sein. Es ist ein natürlicher menschlicher Instinkt, zu einer Gruppe anderer gleichgesinnter Menschen gehören und akzeptiert werden zu wollen. Indem man also Persönlichkeit in ein Web-Framework einbringt und wirklich aktiv mit der Community interagiert, kann diese Art von Gefühl in der Community wachsen.
>
> *– Taylor Otwell, Zitat aus einem Interview der Website »Product and Support«*

Taylor verstand von Anfang an, dass für ein erfolgreiches Open-Source-Projekt zwei Dinge nötig sind: eine gute Dokumentation und eine einladende Community. Und diese beiden Zutaten sind heute Markenzeichen von Laravel.

Wie es funktioniert

Bis jetzt war alles ziemlich abstrakt, zugegeben. Wie wäre es mit ein bisschen Code, fragen Sie vielleicht? Lassen Sie uns in eine einfache Anwendung einsteigen (Beispiel 1-1), damit Sie sehen können, wie die tägliche Arbeit mit Laravel wirklich aussieht.

Beispiel 1-1: »Hello, World« in routes/web.php

```
<?php

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

Die einfachste Aktion, die Sie in einer Laravel-Anwendung durchführen können, besteht darin, eine Route zu definieren und ein Ergebnis zurückzugeben, wenn jemand diese Route aufruft. Wenn Sie eine brandneue Laravel-Anwendung auf Ihrem Computer initialisieren, die Route so definieren wie im Beispiel 1-1 angegeben und dann die Website aus dem Verzeichnis *public* bedienen, haben Sie ein voll funktionsfähiges »Hallo, Welt«-Beispiel (siehe Abbildung 1-1).

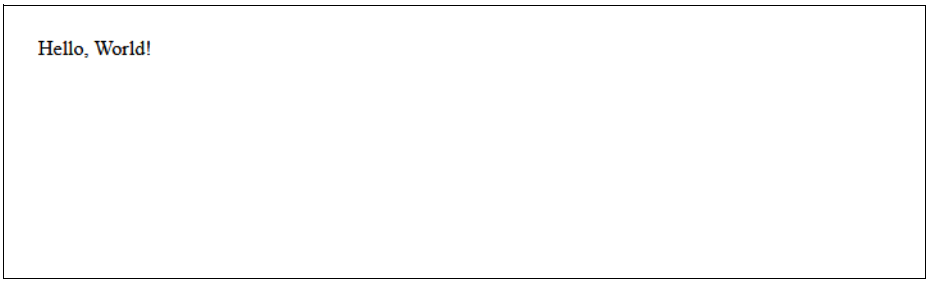

Abbildung 1-1: »Hello, World!« mit Laravel ausgeben

Mit Controllern sieht es ziemlich ähnlich aus, wie Sie in Beispiel 1-2 sehen können.

Beispiel 1-2: »Hello, World« mit Controllern

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

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

// Datei: app/Http/Controllers/WelcomeController.php
<?php

namespace App\Http\Controllers;

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

Und wenn Sie Ihre Grüße an die Welt in einer Datenbank speichern, wird es ebenfalls ziemlich ähnlich aussehen (siehe Beispiel 1-3).

Beispiel 1-3: Mehrfach-Gruß »Hello, World« mit Datenbankzugriff

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

use App\Greeting;

Route::get('create-greeting', function () {
    $greeting = new Greeting;
    $greeting->body = 'Hello, World!';
    $greeting->save();
});

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

```
// Datei: app/Greeting.php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Greeting extends Model
{
    //
}

// Datei: database/migrations/2019_12_19_010000_create_greetings_table.php
<?php

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

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

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

Falls Ihnen Beispiel 1-3 im Moment noch ein wenig kompliziert vorkommt, überspringen Sie es einfach. In den weiteren Kapiteln wird nach und nach alles erklärt, aber Sie können hier bereits sehen, dass Sie mit nur wenigen Programmzeilen Datenbankmigrationen und -modelle einrichten und Datensätze abfragen können. Einfach einfach!

Warum also Laravel?

Weil Laravel Ihnen hilft, Ihre Ideen in die Welt zu bringen, ohne überflüssigen Code produzieren zu müssen, indem Sie moderne Programmierstandards verwenden, unterstützt von einer lebendigen Community und einem leistungsfähigen Ökosystem mit hilfreichen Tools.

Und weil Sie, lieber Entwickler, es verdienen, glücklich und zufrieden zu sein.

KAPITEL 2

Eine Laravel-Entwicklungsumgebung einrichten

Ein Teil des Erfolgs von PHP liegt darin begründet, dass man praktisch keinen Webserver findet, der *kein* PHP ausführen kann. Allerdings haben moderne PHP-Tools strengere Anforderungen als in der Vergangenheit. Wenn man für Laravel entwickelt, ist es am besten, eine konsistente lokale und entfernte Serverumgebung zu gewährleisten – zum Glück bietet das Laravel-Ökosystem für diesen Zweck ein paar Tools an, die dabei helfen können.

Systemanforderungen

Alles, was wir in diesem Kapitel behandeln werden, kann man auch unter Windows einrichten, aber oft muss man dazu leider Dutzende von Seiten mit zusätzlichen Anweisungen und Einschränkungen lesen. Damit wir uns aufs Wesentliche beschränken können, konzentriere ich mich in den Beispielen hier und im Rest des Buchs auf den Einsatz unter Unix/Linux/macOS.

Unabhängig davon, ob Sie PHP und die anderen Tools auf Ihrem lokalen Computer installieren oder die Entwicklungsumgebung in einer virtuellen Maschine über Vagrant oder Docker betreiben oder sich auf ein Tool wie MAMP/WAMP/XAMPP oder Laragon verlassen, müssen in Ihrer Entwicklungsumgebung die folgenden Komponenten installiert sein, damit Laravel-Websites betrieben werden können:

- PHP >= 7.2.0 für Laravel 6.x, PHP >= 7.1.3 für Laravel-Versionen 5.6 bis 5.8, PHP >= 7.0.0 für Version 5.5, PHP >= 5.6.4 für Version 5.4, PHP zwischen 5.6.4 und 7.1.* für Version 5.3, PHP >= 5.5.9 für Versionen 5.2 und 5.1
- OpenSSL (PHP-Erweiterung)
- PDO (PHP-Erweiterung)
- Mbstring (PHP-Erweiterung)
- Tokenizer (PHP-Erweiterung)
- XML (PHP-Erweiterung) für Laravel 5.3 und höher
- Ctype (PHP-Erweiterung) für Laravel 5.6 und höher

- JSON (PHP-Erweiterung) für Laravel 5.6 und höher
- BCMath (PHP-Erweiterung) für Laravel 5.7 und höher

Composer

Egal, auf welchem Rechner auch immer Sie entwickeln, dort muss Composer (*https://getcomposer.org/*) global installiert sein. Falls Sie mit Composer nicht vertraut sind: Es ist ein Werkzeug, das die Grundlage für den Großteil moderner PHP-Entwicklung bildet. Composer ist ein Abhängigkeitsmanager für PHP, ähnlich wie NPM für Node oder RubyGems für Ruby. Aber wie NPM bildet Composer auch die Basis für den überwiegenden Teil unserer Anwendungstests, das Laden lokaler Skripte, Installationsskripte und vieles andere. Sie benötigen Composer, um Laravel zu installieren, es zu aktualisieren und Abhängigkeiten von externen Paketen bzw. Bibliotheken aufzulösen.

Lokale Entwicklungsumgebungen

Für viele Projekte reicht es aus, Ihre Entwicklungsumgebung mithilfe eines eher einfachen Toolsets zu hosten. Wenn Sie bereits MAMP oder WAMP oder XAMPP oder Laragon auf Ihrem System installiert haben, sollte das normalerweise schon ausreichen, um mit Laravel arbeiten zu können. Sie können Laravel auch einfach mit dem integrierten Webserver von PHP ausführen, vorausgesetzt, die auf Ihrem System installierte PHP-Version ist die richtige.

Alles, was Sie wirklich brauchen, ist die Möglichkeit, PHP auszuführen. Alles, was darüber hinausgehen soll, liegt ganz bei Ihnen.

Laravel bietet jedoch zwei Instrumente für die lokale Entwicklung, Valet und Homestead, und diese beide schauen wir uns kurz an. Wenn Sie sich nicht sicher sind, welches Sie verwenden sollen, empfehle ich Ihnen, Valet – auf Deutsch etwa *Kammerdiener* – zu benutzen und sich nur kurz mit Homestead – *Eigenheim* oder *Heimstätte* – vertraut zu machen; beide Werkzeuge sind jedoch wertvoll und nützlich.

Laravel Valet

Wenn Sie den integrierten Webserver von PHP verwenden möchten, ist es am einfachsten, eine Website über eine *localhost*-URL aufzurufen. Wenn Sie `php -S localhost:8000 -t public` im Stammordner Ihrer Laravel-Site ausführen, wird der eingebaute Webserver von PHP Ihre Website unter *[http://localhost:8000/]* anzeigen. Sie können auch den Kommandozeilen-Befehl `php artisan serve` benutzen, nachdem Sie Ihre Anwendung eingerichtet haben, und damit einen gleichwertigen Server starten.

Wenn Sie jedoch Ihre Sites an eine bestimmte Entwicklungsdomain binden möchten, müssen Sie sich mit der Hosts-Datei Ihres Betriebssystems vertraut machen

und ein Tool wie dnsmasq (*https://bit.ly/2eNPJ5T*) verwenden. Lassen Sie uns stattdessen etwas Einfacheres versuchen.

Wenn Sie macOS als Betriebssystem nutzen, kann Ihnen Laravel Valet die Arbeit abnehmen und Ihre Domains mit den Anwendungsordnern verbinden. (Es gibt auch inoffizielle Valet-Forks für Windows und Linux.) Valet installiert dnsmasq und eine Reihe von PHP-Skripten, sodass es danach reicht, `laravel new myapp && open myapp.test` einzugeben, damit alles funktioniert. Sie müssen dann mit Homebrew noch ein paar Tools installieren, aber es sind nur einige wenige, einfache Schritte von der Erstinstallation bis zur Funktionsfähigkeit einer Anwendung.

Installieren Sie Valet – siehe die Dokumentation (*https://bit.ly/2U7uy7b*) für die neuesten Installationsanweisungen – und teilen Sie Valet mit, in welchen Verzeichnissen es nach Ihren Anwendungen suchen soll. Ich führe dazu beispielsweise `valet park` aus meinem Verzeichnis *~/Sites* aus, in dem ich alle meine Anwendungen verwalte, die sich gerade in der Entwicklung befinden. Jetzt können Sie einfach *.test* an das Ende des Verzeichnisnamens anhängen und die URL in Ihrem Browser aufrufen.

Valet bieten verschiedene Optionen: Man kann einfach den Befehl `valet park` benutzen, um auf alle Unterordner eines bestimmten Hauptordners mit *{Ordnername}.test* zuzugreifen; oder man richtet nur einen einzigen Ordner mit `valet link` ein; oder man öffnet eine von Valet verwaltete Domain für einen Ordner mit `valet open`; oder man nutzt für eine Valet-Site HTTPS mit `valet secure` oder öffnet einen ngrok-Tunnel, um die Website mit anderen über `valet share` teilen zu können.

Laravel Homestead

Homestead ist ein weiteres Werkzeug, mit dem Sie eine lokale Entwicklungsumgebung einrichten können. Es ist ein Konfigurationstool, das auf Vagrant aufsetzt (ein Werkzeug zur Verwaltung virtueller Maschinen) und ein vorkonfiguriertes Virtual-Machine-Image bereitstellt, das perfekt für die Laravel-Entwicklung eingerichtet ist *und* die gängige Produktionsumgebung widerspiegelt, auf der viele Laravel-Sites laufen. Homestead ist wahrscheinlich auch die beste lokale Entwicklungsumgebung für Entwickler, die unter Windows arbeiten.

Die Homestead-Dokumentation (*https://bit.ly/2FwQ7EZ*) ist solide und wird ständig auf den neuesten Stand gebracht, deshalb möchte ich Sie dorthin verweisen, wenn Sie wissen möchten, wie Homestead funktioniert und wie Sie es einrichten können.

Vessel

Vessel (*https://vessel.shippingdocker.com/*) ist kein offizielles Laravel-Projekt, aber Chris Fidao von Servers for Hackers (*https://serversforhackers.com/*) und Shipping Docker (*https://shippingdocker.com/*) hat ein einfaches Tool geschrieben, mit dem man Docker-Umgebungen für die Laravel-Entwicklung einrichten kann. Werfen Sie einen Blick in die Dokumentation von Vessel, um mehr zu erfahren.

Ein neues Laravel-Projekt erstellen

Es gibt zwei Möglichkeiten, ein neues Laravel-Projekt zu erstellen – bei beiden geht man über die Befehlszeile. Die erste Option ist die globale Installation des Laravel-Installationsprogramms (mit Composer), die zweite ist die Verwendung der Funktion `create-project` von Composer.

Sie können sich in der Dokumentation der Laravel-Installation (*https://bit.ly/2HFzBFY*) ausführlicher über beide Optionen informieren, aber ich würde das Laravel-Installationsprogramm empfehlen.

Installation von Laravel mit dem Laravel-Installationsprogramm

Wenn Sie Composer global installiert haben, reicht für die Einrichtung des Laravel-Installationsprogramms die Ausführung des folgenden Befehls:

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

Jetzt muss nur noch das Verzeichnis `vendor/bin` der neuen globalen Composer-Installation der Umgebungsvariablen $PATH hinzugefügt werden. Dieses Verzeichnis befindet sich je nach Betriebssystem an verschiedenen Orten, in der Regel finden Sie es hier:

```
// macOS und GNU/Linux-Distributionen:
$HOME/.composer/vendor/bin

// Windows:
%USERPROFILE%\AppData\Roaming\Composer\Vendor\bin
```

Sobald Sie das Laravel-Installationsprogramm eingerichtet haben, ist es sehr leicht, ein neues Laravel-Projekt zu beginnen. Führen Sie einfach diesen Befehl von der Kommandozeile aus:

```
laravel new projectName
```

Dadurch wird ein neues Unterverzeichnis des aktuellen Verzeichnisses namens *{projectName}* erstellt und darin ein »nacktes« Laravel-Projekt initialisiert.

Installation von Laravel mit dem create-project-Feature von Composer

Composer bietet eine Funktion namens `create-project`, mit der man ebenfalls ein neues Projekt mit der vorgegebenen Grundstruktur einrichten kann. Um auf diese Weise ein neues Laravel-Projekt zu erstellen, geben Sie den folgenden Befehl ein:

```
composer create-project laravel/laravel projectName
```

Genau wie beim Installer-Tool wird auch jetzt ein Unterverzeichnis des aktuellen Verzeichnisses namens *{projectName}* angelegt, das ein »Skelett« eines Laravel-Projekts enthält, das sofort zur weiteren Bearbeitung bereit ist.

Das Helfer-Paket installieren

In Laravel gibt es eine große Anzahl von globalen Hilfsfunktionen, die oft auch kurz als *Helfer* bezeichnet werden. Zwei Gruppen dieser Helfer, die mit Zeichenketten und Arrays arbeiten und bis auf wenige Ausnahmen an den Präfixen *str_* und *array_* erkannt werden können, wurden in Version 6 in das neue Composer-Paket `laravel/helpers` verschoben und aus dem Framework entfernt. Stattdessen können diese Hilfsfunktionen jetzt als Methoden der Klassen `Illuminate\Support\Str` und `Illuminate\Support\Arr` aufgerufen werden.

Damit der in diesem Buch gezeigte Beispielcode auch mit früheren Versionen funktioniert, verwende ich vorerst noch die bisherigen Varianten. Installieren Sie deshalb bitte auch das Helfer-Paket:

```
composer require laravel/helpers
```

Mehr zu Hilfsfunktionen finden Sie in Kapitel 17.

Lambo: »laravel new« mit Düsenantrieb

Da ich oft eine Reihe gleicher Schritte ausführen möchte, nachdem ich ein neues Laravel-Projekt erstellt habe, habe ich ein einfaches Skript namens Lambo (*https://bit.ly/2TCcQo8*) geschrieben, um diese Vorgänge zu automatisieren.

Lambo führt `laravel new` aus und überträgt dann Ihren Code an Git, richtet Ihre *.env*-Konfiguration mit vernünftigen Voreinstellungen ein, öffnet das Projekt in einem Browser und (optional) in Ihrem Editor und führt zudem einige weitere hilfreiche Build-Schritte durch.

Sie können Lambo mit dem Composer-Befehl `global require` installieren:

```
composer global require tightenco/lambo
```

Und Sie können es genauso benutzen wie `laravel new`:

```
cd Sites
lambo my-new-project
```

Die Verzeichnisstruktur von Laravel

Wenn Sie ein Verzeichnis öffnen, das das Grundgerüst einer Laravel-Anwendung enthält, sehen Sie die folgenden Dateien und Verzeichnisse:

```
app/
bootstrap/
config/
database/
public/
resources/
routes/
storage/
```

```
tests/
vendor/
.editorconfig
.env
.env.example
.gitattributes
.gitignore
artisan
composer.json
composer.lock
package.json
phpunit.xml
README.md
server.php
webpack.mix.js
```

Verschiedene Build-Tools in Laravel-Versionen vor 5.4

In Projekten, die vor Laravel 5.4 erstellt wurden, werden Sie wahrscheinlich eine Datei *gulpfile.js* anstelle von *webpack.mix.js* sehen; dies zeigt, dass das Projekt Laravel Elixir (*https://bit.ly/2s5aKVm*) anstelle von Laravel Mix (*https://bit.ly/2OTXZGc*) verwendet.

Lassen Sie uns alles nacheinander durchgehen, um uns damit vertraut zu machen.

Die Ordner

Das Stammverzeichnis enthält standardmäßig die folgenden Ordner:

app
: Wird den Großteil der eigentlichen Anwendung enthalten. Modelle, Controller, Befehle und der überwiegende Teil des anwendungsspezifischen PHP-Codes residieren hier.

bootstrap
: Enthält die Dateien, die das Laravel-Framework beim Booten nutzt, wenn es gestartet wird.

config
: Hier befinden sich alle Konfigurationsdateien.

database
: Hier befinden sich die Datenbankmigrationen, Seed-Dateien und Fabriken (Factories).

public
: Das Verzeichnis, auf das der Server zugreift, wenn er die Website ausliefert. Hier liegt auch *index.php*, der Front-Controller, der den Bootstrapping-Prozess startet und alle Anfragen entsprechend weiterleitet. Hier befinden sich auch alle öffentlich zugänglichen Dateien wie Bilder, Stylesheets, Skripte oder Downloads.

resources
: Hier befinden sich Dateien, die von anderen Skripten benötigt werden. Ansichten (Views), Sprachdateien und (optional) Sass/Less/Quell-CSS- und Quell-JavaScript-Dateien.

routes
: Hier befinden sich die Routendefinitionen, sowohl für HTTP-Routen als auch für »Konsolenrouten« oder Artisan-Befehle.

storage
: Die Heimat von Caches, Protokollen und kompilierten Systemdateien.

tests
: Hier residieren die Modul- und Integrationstests.

vendor
: Hier installiert Composer seine »Abhängigkeiten«, also die Module bzw. Pakete, die von anderen Teilen der Anwendung benötigt werden. Der Ordner wird von Git ignoriert, d.h., der Inhalt wird vom Versionskontrollsystem ausgeschlossen, weil davon ausgegangen wird, dass im Rahmen der späteren Bereitstellung der Anwendung auf Remote-Servern ebenfalls Composer ausgeführt wird und dieser sich auch dort alle benötigten Pakete »zieht«.

Die Dateien

Das Stammverzeichnis enthält außerdem die folgenden Dateien:

.editorconfig
: Gibt Ihrem IDE- bzw. Texteditor Anweisungen zu Laravels Codierungsstandards (z.B. zur Größe von Einrückungen, zum Zeichensatz und ob Leerzeichen am Ende einer Zeile entfernt werden sollen). Diese Datei existiert in allen Laravel-Anwendungen ab Version 5.5.

.env und .env.example
: In diesen Dateien werden die Umgebungsvariablen festgelegt (Variablen, die in jeder Umgebung unterschiedlich sein sollen und daher nicht der Versionskontrolle unterliegen). *.env.example* ist eine Vorlage, anhand derer in jeder neuen Umgebung eine eigene *.env*-Datei erstellt werden kann, die dann als »Git-ignored« vermerkt werden sollte.

.gitignore und .gitattributes
: Die Git-Konfigurationsdateien.

artisan
: Erlaubt Ihnen, Artisan-Befehle von der Kommandozeile aus auszuführen (siehe Kapitel 8).

composer.json und composer.lock
: Konfigurationsdateien für Composer; *composer.json* ist benutzerdefiniert und *composer.lock* nicht. Diese Dateien enthalten einige grundlegende Informationen über das Projekt und die Definitionen der PHP-Abhängigkeiten.

package.json und package-lock.json
: Wie *composer.json* und *composer.lock*, aber für Frontend-Assets und Abhängigkeiten des Build-Systems; es weist NPM an, welche JavaScript-basierten Abhängigkeiten einzubeziehen sind.

phpunit.xml
: Eine Konfigurationsdatei für PHPUnit, das Tool, das Laravel zum Testen verwendet.

README.md
: Eine Datei im Markdown-Format, die eine grundlegende Einführung in Laravel bietet. Diese Datei entfällt, wenn Sie den Laravel-Installer verwenden.

server.php
: Ein Backup-Server, der versucht, Servern mit geringerer Funktionalität dennoch eine Vorschau der Laravel-Anwendung zu ermöglichen.

webpack.mix.js
: Die (optionale) Konfigurationsdatei für Mix. Wenn Sie Elixir verwenden, gibt es stattdessen die Datei *gulpfile.js*. Diese Dateien dienen dazu, Ihrem Build-System Anweisungen zur Kompilierung und Verarbeitung Ihrer Frontend-Assets zu geben.

Konfiguration

Die zentralen Einstellungen Ihrer Laravel-Anwendung – zu Datenbankverbindungen, Warteschlangen und zur E-Mail-Konfiguration u. v. m. – werden in Dateien im Ordner *config* gespeichert. Jede dieser Dateien gibt ein PHP-Array zurück, und jeder Wert in einem solchen Array ist über einen Konfigurationsschlüssel zugänglich, der sich aus dem Dateinamen und den untergeordneten Schlüsseln zusammensetzt, getrennt durch Punkte (`.`).

Wenn Sie also beispielsweise in der Datei *config/services.php* folgenden Eintrag hätten ...

```
// config/services.php
<?php
return [
    'third_party_service' => [
        'secret' => 'abcdefg',
    ],
];
```

... könnten Sie auf diese Konfigurationsvariable mit `config('services.third_party_service.secret')` zugreifen.

Konfigurationsvariablen, die sich je nach Umgebung – z. B. in Entwicklung und Produktion – unterscheiden (und daher nicht der Versionsverwaltung unterliegen sollten) werden stattdessen in einer *.env*-Datei definiert. Nehmen wir an, Sie möchten für jede Umgebung einen eigenen Bugsnag-API-Schlüssel verwenden. Dann

würden Sie es in der Konfigurationsdatei so formulieren, damit dieser Schlüssel aus *.env* eingelesen wird:

```
// config/services.php
<?php
return [
    'bugsnag' => [
        'api_key' => env('BUGSNAG_API_KEY'),
    ],
];
```

Die `env()`-Helferfunktion liest einen Wert aus der aktuellen *.env*-Datei, der zum angegebenen Schlüssel gehört. Fügen Sie also jetzt diesen Schlüssel *mit* Wert Ihrer *.env*-Datei hinzu (also den Einstellungen für die aktuelle Umgebung) und ebenso der *.env.example*-Datei (der Vorlage für alle Umgebungen), hier aber *ohne* Wert:

```
# In .env
BUGSNAG_API_KEY=oinfp9813410942

# In .env.example
BUGSNAG_API_KEY=
```

Ihre *.env*-Datei enthält bereits einige umgebungsspezifische Variablen, die vom Framework benötigt werden, z.B. die Angabe, welcher Mail-Treiber verwendet werden soll und welche grundlegenden Datenbankeinstellungen es gibt.

env() außerhalb von Konfigurationsdateien

Bestimmte Funktionen in Laravel, einschließlich einiger Caching- und Optimierungsfeatures, sind nicht verfügbar, wenn `env()`-Aufrufe außerhalb von Konfigurationsdateien verwendet werden.

Der beste Weg, Umgebungsvariablen verfügbar zu machen, besteht darin, Konfigurationselemente für solche Einstellungen anzulegen, die umgebungsspezifisch sein sollen. In diesen Konfigurationselementen lesen Sie die Umgebungsvariablen mit `env()` ein und können dann per `config()` an beliebiger Stelle in Ihrer Anwendung auf diese Konfigurationsvariablen verweisen:

```
// config/services.php
return [
    'bugsnag' => [
        'key' => env('BUGSNAG_API_KEY'),
    ],
];

// In einem Controller oder an beliebiger Stelle in der
// Anwendung
$bugsnag = new Bugsnag(config('services.bugsnag.key'));
```

Die .env-Datei

Lassen Sie uns jetzt einen kurzen Blick auf den Standardinhalt der Datei *.env* werfen. Die genauen Schlüssel variieren je nach verwendeter Version von Laravel, aber in Beispiel 2-1 sehen Sie die Schlüssel, wie sie in Version 6.6 aussehen.

Beispiel 2-1: Standardwerte der Umgebungsvariablen in Laravel 6.6

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

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
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

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

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

Ich werde nicht auf alle eingehen, denn viele davon sind reine Authentifizierungsinformationen für verschiedene Dienste (Pusher, Redis, Datenbank, E-Mail). Hier folgen jedoch zwei wichtige Umgebungsvariablen, die Sie kennen sollten:

`APP_KEY`
: Eine zufällig generierte Zeichenkette, die zur Verschlüsselung von Daten verwendet wird. Wenn dieser Schlüssel leer ist, könnten Sie die Fehlermeldung »No application encryption key has been specified.« (»Es wurde kein Anwen-

dungsverschlüsselungscode angegeben.«) erhalten. In diesem Fall führen Sie einfach den Befehl `php artisan key:generate` aus, und Laravel wird einen solchen Schlüssel für Sie generieren.

`APP_DEBUG`
: Ein boolescher Wert, der festlegt, ob Benutzer Ihrer Anwendung Debug-Meldungen sehen sollen – wunderbar geeignet in lokalen und Staging-, katastrophal in Produktionsumgebungen.

Der Rest der Einstellungen, der keinen Authentifizierungszwecken dient (`BROADCAST_DRIVER`, `QUEUE_CONNECTION` usw.), enthält Standardwerte, die mit möglichst wenig Abhängigkeit von externen Services arbeiten, was zu Beginn ideal ist.

Wenn Sie mit einer Laravel-Anwendung loslegen, ist die einzige Änderung, die Sie wahrscheinlich für die meisten Projekte vornehmen werden, eine Änderung der Datenbank-Konfigurationseinstellungen. Ich benutze Laravel Valet, also ändere ich `DB_DATABASE` auf den Namen meines Projekts, `DB_USERNAME` auf `root` und `DB_PASSWORD` auf eine leere Zeichenkette:

```
DB_DATABASE=myProject
DB_USERNAME=root
DB_PASSWORD=
```

Dann erstelle ich in meinem bevorzugten MySQL-Client eine Datenbank, die den gleichen Namen wie mein Projekt erhält, und schon bin ich startbereit.

Achtung, fertig, los!

Jetzt ist Ihre – noch »leere« – Laravel-Installation nahezu startklar. Führen Sie `git init` aus, committen Sie die Projektdateien mit `git add .` und `git commit`, und schon ist alles bereit, um mit der Programmierung beginnen zu können. So einfach ist das! Und falls Sie Valet verwenden, können Sie die folgenden Befehle ausführen und Ihre Website sofort live in Ihrem Browser sehen:

```
laravel new myProject && cd myProject && valet open
```

Jedes Mal, wenn ich ein neues Projekt beginne, sind dies also die Schritte, die ich ausführe:

```
laravel new myProject
cd myProject
git init
git add .
git commit -m "Initial commit"
```

Ich verwalte alle meine Websites in einem Ordner *~/Sites*, den ich als mein primäres Valet-Verzeichnis eingerichtet habe, sodass ich in diesem Fall in meinem Browser sofort und ohne weitere Vorbereitung *myProject.test* aufrufen kann. Ich kann die *.env*-Datei bearbeiten und auf eine bestimmte Datenbank verweisen, diese in meinem MySQL-Client anlegen und bin bereit, mit der Programmierung zu begin-

nen. Und denken Sie daran: Wenn Sie Lambo verwenden, können Sie alle diese Schritte automatisiert ausführen.

Testen

In jedem der weiteren Kapitel zeige ich Ihnen zum Abschluss in einem Abschnitt »Testen«, wie Sie Tests für die beschriebenen Features anlegen können. Da dieses Kapitel aber kein testfähiges Feature behandelt, lassen Sie uns kurz grundlegend über Tests sprechen. (Um mehr über das Schreiben und Ausführen von Tests in Laravel zu erfahren, gehen Sie bitte zu Kapitel 12.)

Laravel bringt PHPUnit als installierte Abhängigkeit mit und ist so konfiguriert, dass Tests aus all jenen Dateien im Verzeichnis *tests* ausgeführt werden, deren Name mit *Test.php* endet (z. B. *tests/UserTest.php*).

Der einfachste Weg, Tests anzulegen, besteht also darin, eine Datei im Verzeichnis *tests* mit einem Namen zu erstellen, der mit *Test.php* endet. Und am einfachsten führt man diese Tests mit dem Befehl `./vendor/bin/phpunit` von der Kommandozeile (im Rootverzeichnis des Projekts) aus.

Wenn Tests einen Datenbankzugriff erfordern, führen Sie Ihre Tests auf dem Rechner aus, auf dem auch Ihre Datenbank liegt – wenn Sie also Ihre Datenbank in Vagrant hosten, stellen Sie sicher, dass Sie per `ssh` auf Ihre Vagrant-Box zugreifen, um Ihre Tests von dort aus auszuführen. Auch dazu und zu weiteren Aspekten können Sie mehr in Kapitel 12 erfahren.

In einigen der folgenden Kapitel können in den Abschnitten zum Testen auch Syntaxelemente und Funktionen vorkommen, mit denen Sie noch nicht vertraut sind, wenn Sie das Buch zum ersten Mal lesen. Wenn Sie Code in einem dieser Abschnitte verwirrend finden, überspringen Sie ihn einfach und kehren noch einmal zurück, wenn Sie das Kapitel über Tests gelesen haben.

TL;DR

Da Laravel ein PHP-Framework ist, lässt es sich sehr einfach lokal ausführen. Laravel bietet zudem zwei Werkzeuge für die Verwaltung Ihrer lokalen Entwicklungsumgebung: ein einfacheres Tool namens Valet, das die Abhängigkeiten auf Ihrem lokalen Computer installiert, sowie ein vorkonfiguriertes Vagrant-Setup namens Homestead. Laravel basiert auf und kann von Composer installiert werden und bringt von Haus aus eine Reihe von Ordnern und Dateien mit, die sowohl die eigenen Konventionen als auch die Beziehungen zu anderen Open-Source-Tools widerspiegeln.

KAPITEL 3

Routing und Controller

Die wesentliche Funktion eines Frameworks für Webanwendungen besteht darin, Anfragen eines Benutzers entgegenzunehmen und diesem Antworten zu liefern, in der Regel über HTTP(S). Deshalb muss man als Allererstes herausfinden, wie man in einer Anwendung Routen definiert, wenn man anfängt, sich mit einem Web-Framework zu beschäftigen; ohne Routen haben Sie wenig bis gar keine Möglichkeiten, überhaupt mit dem Endbenutzer zu interagieren.

In diesem Kapitel werden wir uns mit dem Routen-Handling in Laravel befassen; Sie werden erfahren, wie man Routen definiert, wie man sie mit dem auszuführenden Code verknüpft und wie man die Routing-Tools darüber hinaus verwenden kann, um eine Vielzahl weiterer Aufgaben zu erfüllen.

Eine kurze Einführung in MVC, HTTP-Verben und REST

Dieses Kapitel beschäftigt sich überwiegend mit der Struktur von Model-View-Controller-Anwendungen, und in vielen Beispielen kommen REST-artige Routennamen und Verben vor – lassen Sie uns deshalb auf beide einen kurzen Blick werfen.

Was ist MVC?

In MVC, dem Model-View-Controller-Entwurfsmuster, gibt es drei grundlegende Konzepte:

Model
: Stellt eine einzelne Datenbanktabelle (oder einen Datensatz aus dieser Tabelle) dar – denken Sie an eine »Firma« oder einen »Hund«.

View
: Bezeichnet die Ansicht bzw. Vorlage, die Ihre Daten an den Endbenutzer ausgibt – z. B. eine »Log-in-Seite mit HTML-, CSS- und JavaScript-Code«.

Controller

Der Controller nimmt HTTP-Anfragen vom Browser entgegen, besorgt die richtigen Daten aus der Datenbank oder anderen Speichern, validiert Benutzereingaben und sendet schließlich eine Antwort zurück an den Benutzer.

In Abbildung 3-1 können Sie sehen, dass der Endbenutzer zuerst mit dem Controller interagiert, wenn er über seinen Browser eine HTTP-Anfrage sendet. Als Reaktion auf diese Anforderung schreibt der Controller Daten in das Modell bzw. die Datenbank und/oder liest Daten aus diesem bzw. dieser ein. Der Controller sendet dann wahrscheinlich Daten an eine View, woraufhin diese Ansicht an den Endbenutzer zurückgegeben wird, um sie in seinem lokalen Browser anzuzeigen.

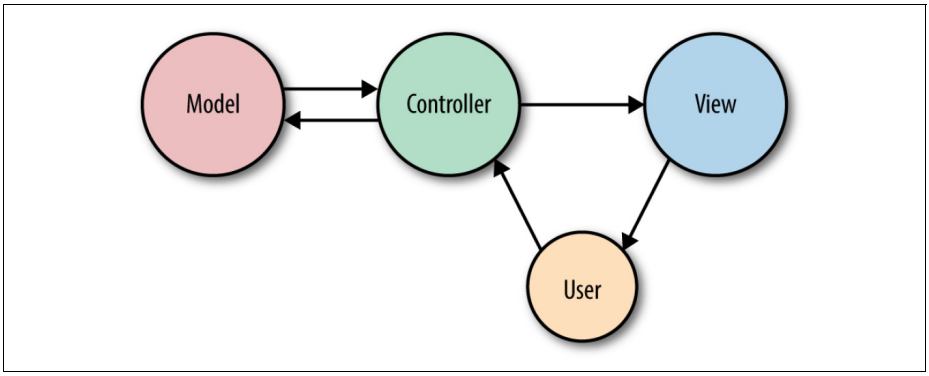

Abbildung 3-1: Eine grundlegende Darstellung von MVC

Wir werden uns im weiteren Verlauf auch einige Anwendungsfälle in Laravel anschauen, die nicht zu dieser relativ vereinfachten Betrachtungsweise der Anwendungsarchitektur passen. Betrachten Sie das MVC-Modell bitte nicht als absolut – es geht jetzt mehr darum, uns auf einen späteren Abschnitt dieses Kapitels vorzubereiten, in dem es um Views und Controller gehen wird.

Die HTTP-Verben

Die gebräuchlichsten HTTP-Verben sind `GET` und `POST`, gefolgt von `PUT` und `DELETE`. Es gibt auch noch `HEAD`, `OPTIONS` und `PATCH` sowie zwei weitere, die in der normalen Webentwicklung so gut wie nie verwendet werden: `TRACE` und `CONNECT`.

Hier ein kurzer Überblick:

`GET`

Anfordern einer Ressource (oder einer Liste von Ressourcen).

`HEAD`

Fragt nach einer reinen Header-Version der Antwort, die auf eine `GET`-Anfrage geliefert wird.

`POST`

Erstellt eine Ressource.

`PUT`
: Überschreibt eine Ressource.

`PATCH`
: Ändert eine Ressource.

`DELETE`
: Löscht eine Ressource.

`OPTIONS`
: Fragt den Server, welche HTTP-Verben bei dieser URL erlaubt sind.

Tabelle 3-1 zeigt die Aktionen, die mit einem Ressourcen-Controller verfügbar sind (mehr dazu unter »Ressourcen-Controller« auf Seite 47). Jede Aktion wird durch den Aufruf eines bestimmten URL-Musters mit einem bestimmten HTTP-Verb ausgelöst. In der Übersicht können Sie – am Beispiel einer Aufgabenplanung – einen Eindruck davon gewinnen, wofür jedes Verb verwendet wird.

Tabelle 3-1: HTTP-Verben und zugeordnete Methoden in Ressourcen-Controllern

Verb	URL	Controller-Methode	Name	Beschreibung
`GET`	`tasks`	`index()`	`tasks.index`	Alle Aufgaben (Tasks) anzeigen
`GET`	`tasks/create`	`create()`	`tasks.create`	Das Formular für die Erstellung der Aufgabe anzeigen
`POST`	`tasks`	`store()`	`tasks.store`	Die Daten aus dem Formular übergeben und speichern
`GET`	`tasks/{task}`	`show()`	`tasks.show`	Eine Aufgabe anzeigen
`GET`	`tasks/{task}/edit`	`edit()`	`tasks.edit`	Eine Aufgabe bearbeiten
`PUT/PATCH`	`tasks/{task}`	`update()`	`tasks.update`	Geänderte Daten aus dem Formular übergeben und speichern
`DELETE`	`tasks/{task}`	`destroy()`	`tasks.destroy`	Eine Aufgabe löschen

Was ist REST?

Wir werden REST in »Die Grundlagen REST-ähnlicher JSON-APIs« auf Seite 337 ausführlicher behandeln, aber es ist, kurz gesagt, ein Architekturmuster zum Erstellen von APIs. Wenn wir in diesem Buch über REST sprechen, beziehen wir uns hauptsächlich auf einige Merkmale wie z. B.:

- Strukturierung um jeweils genau eine Primärressource (z. B. `tasks`)
- Bestehend aus Interaktionen mit voraussagbaren bzw. planbaren URL-Strukturen unter Verwendung von HTTP-Verben (wie in Tabelle 3-1 zu sehen)
- Rückgabe von JSON und oft auch angefordert mit JSON

Das ist nur eine Auswahl – wenn wir den Begriff »REST-artig« in diesem Buch verwenden, ist damit eine »URL-basierte Aufrufstruktur« gemeint, sodass wir plan-

bare Aufrufe wie `GET /tasks/14/edit` ausführen können, um beispielsweise eine Bearbeitungsseite für eine bestimmte Aufgabe aufzurufen. Dies ist auch relevant, wenn es nicht um APIs geht, da das Routing in Laravel grundsätzlich auf einer REST-ähnlichen Struktur basiert, wie Sie in Tabelle 3-1 sehen können.

REST-basierte APIs folgen hauptsächlich derselben Struktur, abgesehen davon, dass sie keine *create*- oder *edit*-Route aufweisen, denn APIs führen nur Aktionen aus, aber zeigen keine Seiten an, auf denen diese Aktionen vorbereitet werden.

Routendefinitionen

In einer Laravel-Anwendung definieren Sie Webrouten in *routes/web.php* und API-Routen in *routes/api.php*. Webrouten sind diejenigen, die von Ihren Endbenutzern besucht werden; API-Routen sind diejenigen für Ihre API, falls Sie eine anbieten. Im Moment konzentrieren wir uns vor allem auf die Routen in *routes/web.php*.

Speicherort der Routendatei in Laravel-Versionen vor 5.3
In Versionen vor 5.3 gibt es nur *eine* Routendatei, und zwar *app/Http/routes.php*.

Die einfachste Art, eine Route zu definieren, besteht darin, einem bestimmten Pfad (z.B. `/`) eine Closure zuzuordnen wie in Beispiel 3-1.

Beispiel 3-1: Grundlegende Routendefinition

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

Was ist eine Closure?

Eine Closure, auch als Funktionsabschluss bezeichnet, ist die PHP-Version einer anonymen Funktion. Eine Closure lässt sich einer Variablen zuweisen, als Parameter an andere Funktionen und Methoden oder als Objekt übergeben und sogar serialisieren.

Sie haben nun festgelegt, dass bei einem Aufruf des Stammverzeichnisses Ihrer Domain die definierte Closure ausgeführt und das Ergebnis dieses Funktionsaufrufs zurückgegeben werden soll. Beachten Sie, dass wir unseren Inhalt mit `return` tatsächlich *zurückgeben* und keine `echo`- oder `print`-Anweisung benutzen.

Was ist Middleware?

Sie fragen sich vielleicht: »Warum gebe ich *Hello, World!* mit `return` zurück und nicht mit `echo` aus?«

Darauf gibt es verschiedene Antworten, aber die einfachste lautet, dass es viele Wrapper gibt, die Laravels Request/Responce Cycle umschließen (den Zyklus von eingehender Anfrage und ausgehender Antwort), u. a. auch die sogenannte *Middleware*. Wenn der Aufruf einer Routen-Closure oder einer Controller-Methode abgeschlossen ist, wird die Antwort nicht direkt an den Browser gesendet. Weil der Inhalt mit `return` zurückgegeben wird, kann er den weiteren Response-Stack und die gesamte Middleware durchlaufen und darin weiter verarbeitet oder »veredelt« werden, bevor er endgültig an den Benutzer zurückgegeben wird.

Viele einfache Websites können vollständig in der Webroutendatei definiert werden. Mit ein paar einfachen `GET`-Routen in Kombination mit einigen Vorlagen, wie in Beispiel 3-2 veranschaulicht, können Sie sehr leicht eine klassische Website aufsetzen.

Beispiel 3-2: Einfache Beispiel-Website

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

Route::get('about', function () {
    return view('about');
});

Route::get('products', function () {
    return view('products');
});

Route::get('services', function () {
    return view('services');
});
```

Statische Aufrufe

Wenn Sie viel Erfahrung in der Entwicklung mit PHP haben, werden Sie vielleicht überrascht sein, statische Aufrufe der Klasse `Route` zu sehen. Das sind allerdings keine statischen Methodenaufrufe per se, sondern eine Art Dienstleistung, die von Laravels Fassaden erbracht wird und die wir uns in Kapitel 11 genauer anschauen werden.

Wenn Sie keine Fassaden benutzen möchten, können Sie die Definitionen auch wie folgt anlegen:

```
$router->get('/', function () {
    return 'Hello, World!';
});
```

Routing-Verben

Vielleicht haben Sie bemerkt, dass wir `Route::get()` in unseren Routendefinitionen verwendet haben. Damit weisen wir Laravel an, diese Routen nur aufzurufen, wenn der HTTP-Request die `GET`-Methode verwendet. Aber was passiert, wenn es sich um Formulardaten handelt, die mit `POST` gesendet werden, oder vielleicht um ein JavaScript, das `PUT`-oder `DELETE`-Anfragen sendet? Auch dafür gibt es passende Methoden, die man bei Routen einsetzen kann, wie Sie in Beispiel 3-3 sehen können.

Beispiel 3-3: Routing-Verben

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

Route::post('/', function () {
    // Wird ausgeführt, wenn jemand eine POST-Anfrage an diese Route sendet
});

Route::put('/', function () {
    // Wird ausgeführt, wenn jemand eine PUT-Anfrage an diese Route sendet
});

Route::delete('/', function () {
    // Wird ausgeführt, wenn jemand eine DELETE-Anfrage an diese Route sendet
});

Route::any('/', function () {
    // Wird ausgeführt, egal welche HTTP-Methode für eine Anfrage an diese Route
    // verwendet wird
});

Route::match(['get', 'post'], '/', function () {
    // Wird ausgeführt, wenn GET- oder POST-Anfragen an diese Route gesendet werden
});
```

Routen-Handling

Wie Sie wahrscheinlich schon erraten haben, ist die Angabe einer Closure nicht der einzige Weg, eine Route zu definieren. Closures sind eine schnelle Lösung, aber je größer eine Anwendung wird, desto nachteiliger ist es, die gesamte Routinglogik in einer einzigen Datei zu speichern. Darüber hinaus können Anwendungen, die Closures in Routendefinitionen verwenden, Laravels Routen-Caching (mehr dazu später) nicht nutzen, das bei jeder Anfrage einige Hundert Millisekunden einsparen kann.

Die andere gängige Option ist deshalb, anstelle einer Closure eine bestimmte Controller-Methode anzugeben wie in Beispiel 3-4.

Beispiel 3-4: Routen, die Controller-Methoden aufrufen

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