



Niclas Kahlmeier

Laravel 7

Ein Webmasters Press Lernbuch

Version 1.0.4 vom 06.07.2020

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

Inhaltsverzeichnis

1	Einleitung	11
1.1	An wen richtet sich diese Class und was lerne ich?	11
1.2	Warum Laravel?	11
1.3	Voraussetzungen	12
1.4	Aufbau der Class	13
2	Wiederholung einiger wichtiger Grundlagen	15
2.1	Exkurse: Git und Linux	15
2.1.1	Unix/Linux-Exkurs	15
2.1.2	Git-Exkurs	16
2.2	Objektorientiertes PHP + PHP 7.4	16
2.2.1	Clean Code & SOLID	16
2.2.2	Generator – iteriere über große Datensätze ohne Probleme mit dem Arbeitsspeicher	17
2.2.3	Inheritance	17
2.2.4	Traits	18
2.2.5	Abstracts	18
2.2.6	Interfaces	20
2.2.7	Rekursion	20
2.2.8	PHP 7.4	21
3	Basisarchitektur von Laravel	23
3.1	MVC in Laravel	23
3.2	Warum verwendet man die MVC-Architektur?	23
3.3	Life Cycle	24
3.4	Namenskonventionen (Naming Conventions)	25
3.5	Teste dein Wissen!	26
4	Homestead - Virtualisierung der Entwicklungsumgebung	28
4.1	Warum benutzen die meisten Laravel-Entwickler Homestead?	28
4.2	Installation	29
4.3	Die Dateistruktur von Laravel	33
4.4	Teste dein Wissen!	36
5	Routing-Grundlagen	37
5.1	Routes benennen	39
5.2	Weiterleitung	39
5.3	Controller Routes	39
5.4	Teste dein Wissen!	40
6	PHP Artisan – viele nützliche Befehle	42
7	Controller	44
7.1	Wozu sind Controller gut?	44
7.2	Wie werden Controller erstellt?	44
7.3	Controller mit Artisan Commands erstellen	45

7.4	Resource Controller	46
7.5	Teste dein Wissen!	48
8	Facades, Service Container & Contracts – Grundkonzepte von Laravel	50
8.1	Service Container = IoC Container	50
8.2	Facades	52
8.3	Contracts	53
8.4	Closures	54
8.5	Teste dein Wissen!	54
9	Request und Response – zu jeder Frage eine Antwort	56
9.1	Request – Anfrage des Browsers an Laravel	56
9.1.1	Zugriff auf Adresse	57
9.1.2	Zugriff auf übergebene Werte	57
9.2	Response	58
10	Views	63
10.1	Datenübermittlung an die View	64
11	Blade Templates	66
11.1	Daten wiedergeben	66
11.2	Blade-Kontrollstrukturen	68
11.2.1	If-Abfragen	68
11.2.2	Switch-Abfragen	69
11.2.3	Schleifen	69
11.2.4	Inline-PHP	69
11.2.5	Kommentare	69
11.3	Template Inheritance	72
11.4	Include	74
11.5	Grafik zum Veranschaulichen	75
11.6	Stacks	75
11.7	Komponenten einbinden	75
11.8	Blade Component Tags	77
11.8.1	Anonyme Komponenten	78
12	Datenbanken	81
12.1	Mehrere Datenbankverbindungen	82
12.2	Teste dein Wissen!	82
13	Migrations – das Git für Datenbanken	83
13.1	Struktur der Migration	84
13.2	Schema Builder	85
13.2.1	Tabellen erstellen	85
13.2.2	Tabellen bearbeiten	85
13.2.3	Tabellenspalte erstellen	85
13.2.4	Column Modifier	86
13.2.5	Tabellenspalten bearbeiten & löschen	87
13.2.6	Existenz von Tabelle und Tabellenspalte prüfen	87
13.2.7	Spezifische Datenbankverbindung im Schema nutzen	88
13.2.8	Tabellen umbenennen und löschen	88
13.2.9	Indexe	88

13.3	Migration ausführen/migrieren	89
13.4	Teste dein Wissen!	90
14	Raw Queries	93
14.1	Insert – speichere Daten in der Datenbank	93
14.2	Select – lies Datenbankinformationen	93
14.3	Update – aktualisiere Datenbankinformationen	94
14.4	Delete – Daten löschen	94
14.5	Statement	94
14.6	Teste dein Wissen!	95
15	Query Builder – Datenbankabfragen ohne SQL schreiben	96
15.1	Debugging	96
15.2	Insert – Inhalte einfügen	97
15.3	Query Chaining	100
15.4	Daten abrufen	101
15.5	Select – Daten auswählen	103
15.6	where-Abfragen – Bedingungen festlegen	103
15.6.1	where-Abfragen in JSON	104
15.6.2	where-Closures	105
15.7	when-Abfragen	106
15.8	Inhalte aktualisieren – Update	106
15.9	Inhalte löschen – Delete	107
15.10	Datenbankwerte sortiert ausgeben	107
15.11	Chunking	108
15.12	Aggregatfunktionen – Abfrageergebnisse zusammenfassen und auswerten	108
15.13	Joins	109
15.14	Unions – Queries vereinen	109
15.15	Raw Expressions	111
15.16	Teste dein Wissen!	111
16	Eloquent ORM	114
16.1	Eloquent Model	115
16.1.1	CRUD mit dem Model	115
16.1.2	Query Scope	120
16.1.3	Weitere Definitionen im Model	120
16.2	Eloquent-Beziehungen	121
16.2.1	One To One	121
16.2.2	One To Many	123
16.2.3	Many to Many	124
16.2.4	Pivot-Tabelle	125
16.2.5	Has One Through	125
16.2.6	Has Many Through	126
16.2.7	Tinker	126
16.2.8	Polymorphic Relationships	127
16.2.9	In Beziehung stehende Models hinzufügen und bearbeiten	133
16.2.10	Beziehungen durchsuchen und Inhalte abfragen	134
16.3	Accessors & Mutators - bearbeiten von Eigenschaften	138
16.3.1	Accessor	139
16.3.2	Mutator	139

17	Collections	144
17.1	Lazy Collections	146
17.2	Eloquent Collections	146
18	Formulare	148
19	Validierung	151
19.1	Teste dein Wissen!	153
20	Sicherheit	155
20.1	CSRF Protection	155
20.2	XSS-Angriff	155
20.3	SQL Injection	156
20.4	Verschlüsselung	156
20.4.1	Encrypter	157
20.4.2	Hashing	157
21	Seeding und Factories	159
22	Session und Cookies	164
23	Deploy Application	168
24	Übung zur Vorbereitung auf die Einsendeaufgabe	169
	Lösungen der Übungsaufgaben	171
	Lösungen der Wissensfragen	177
	Index	196

Basisarchitektur von Laravel

3

Du lernst in dieser Lektion

- die Architektur und den Arbeitsablauf von Laravel kennen.
- wie die MVC-Architektur in Laravel umgesetzt ist.
- welche Namenskonventionen in Laravel üblich sind.

Bevor es mit dem Programmieren richtig losgeht, gebe ich dir einen kurzen Überblick über die Architektur von Laravel. Du wirst jetzt vielleicht mit den Augen rollen ... ;-) Ich denke aber, es wird dir dabei helfen, Struktur und Konzept von Laravel zu verstehen.

3.1 MVC in Laravel

Die Model-View-Controller-Architektur, kurz MVC, teilt die Applikation in die drei namensgebenden Teile:

Das **Model** repräsentiert alle Daten und die Datenverwaltung der Anwendung. Es antwortet außerdem auf Informationsanfragen und reagiert auf Anforderungen, Daten zu ändern. Es registriert alle Datenänderungen und benachrichtigt, sobald sich Informationen ändern.

Die **View** repräsentiert die Benutzeroberfläche. Sie rendert auch die Daten aus dem Model in eine Form, die für die Benutzeroberfläche geeignet ist. Außerdem erhält die View Nutzer-Input und verarbeitet diesen oder leitet ihn weiter.

Der **Controller** ist die Schaltzentrale. Er erhält die Nutzeranfragen und stellt beim Model Anfragen nach Daten. Die entsprechenden Modelobjekte und Views werden aufgerufen und die notwendigen Daten übergeben.

Merksatz: Das Model steuert die Datenbank. Die View kümmert sich um das HTML. Der Controller ist der »Mittelsmann« zwischen Model und View.



3.2 Warum verwendet man die MVC-Architektur?

Die erste Frage, die du dir vielleicht gestellt hast, ist: »Warum macht man das überhaupt?« Wenn man eine kleine Codebase hat, ist es noch nicht notwendig. Bei größeren Projekten ist es jedoch enorm wichtig, den Überblick zu behalten. Die MVC-Architektur verleiht dem Code mehr Struktur und erleichtert dadurch die Arbeit.

Die MVC-Architektur hat weitere Vorteile:

- Die Rollenverteilung im Team wird einfacher. Frontend-Entwickler kümmern sich um Views und Backend-Entwickler um Controller und Models.
- MVC zwingt Entwickler, ihre Dateien in logische Verzeichnisse zu unterteilen. Dadurch lassen sich Dateien und auch ihre Funktionen leichter erkennen und finden.
- Die Models und Views können separat voneinander bearbeitet werden, da diese nicht direkt voneinander abhängig sind. Dieses Prinzip nennt man **Separation of Concerns** (auf Deutsch in etwa »Trennung der Zuständigkeiten«).

3.3 Life Cycle

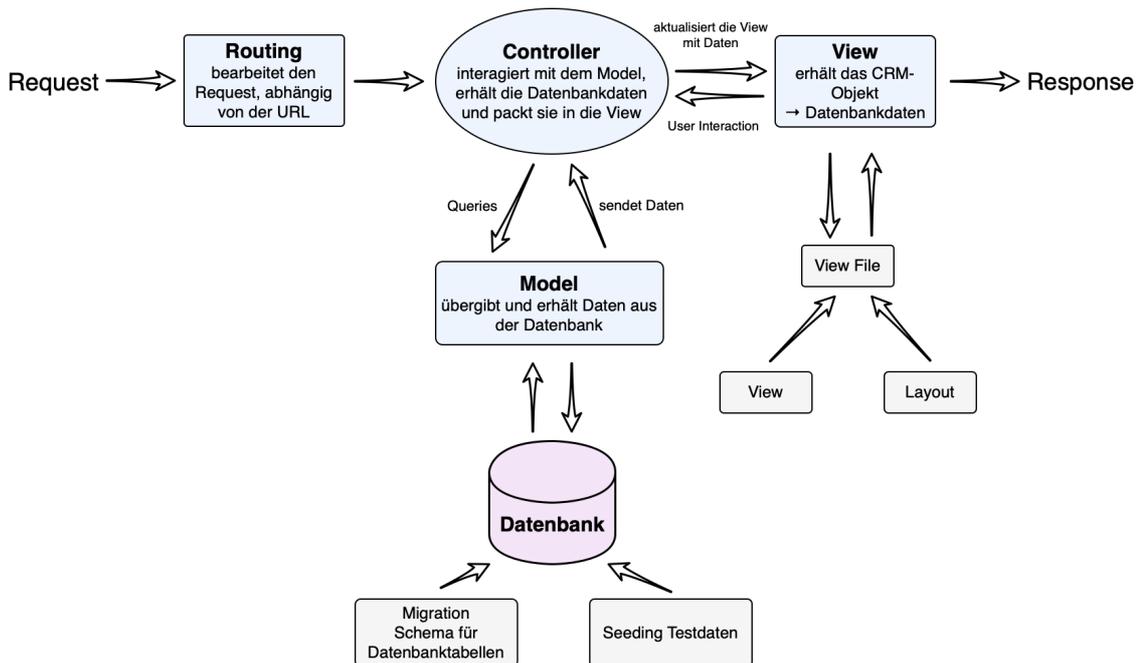


Abb. 1 Workflow in Laravel

In [Abb. 1](#) siehst du eine Grafik zum Workflow: Der Aufruf einer Webseite, die eine eindeutige Adresse im Web hat (Uniform Resource Locator, URL) geschieht mit Hilfe des HTTP-Protokolls (HTTP Request). Die Anfrage wird dann im Routing verarbeitet. Abhängig von der URL wird der entsprechende Controller aufgerufen. Der Controller kommuniziert mit der Datenbank, um z. B. Daten daraus abzurufen oder Daten in die Datenbank zu schreiben. Das Model leitet die entsprechenden Daten an den Controller weiter. Der Controller ruft die entsprechende View auf und gibt die Daten weiter. Die View kann sich auch aus mehreren Dateien zusammensetzen. Nachdem die View die notwendigen Daten erhalten hat, wird sie gerendert. Nach dem Rendern wird eine Antwort (HTTP Response) an den Webbrowser gesendet. Dies ist dann die für den Nutzer sichtbare Webseite.

Die gerade schon besprochene MVC-Architektur habe ich in der Grafik ([Abb. 2](#)) rot markiert.

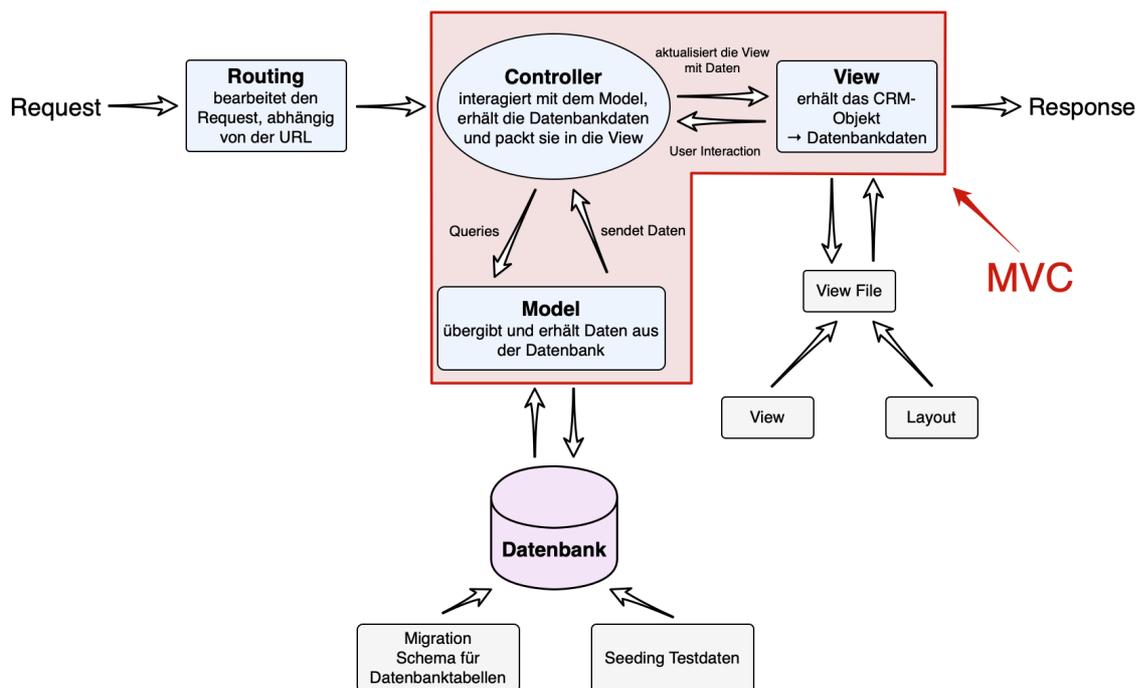


Abb. 2 MVC-Architektur innerhalb des Workflow

3.4 Namenskonventionen (Naming Conventions)

Laravel wird von Tausenden Entwicklern für die verschiedensten Websites und Applikationen genutzt. Dabei orientieren sie sich an etablierten Regeln zur Namensgebung von Funktionen, Klassen und Variablen. Dies hilft anderen Entwickler dabei, sich in dem von anderen geschriebenen Code zurechtfinden. Ich gehe im Folgenden die verschiedenen Bezeichnungen durch. Vielleicht sagen dir die genannten Begriffe jetzt noch nichts, aber wir werden diese im weiteren Verlauf alle kennenlernen.

Bezeichnung von Controllern

Namen für Controller werden im Singular und großgeschrieben, enthalten keine Leerzeichen und enden auf *Controller*.

Beispiel: *UserController*, *AuthController*, *RoleController*

Diese Schreibweise nennt man **Pascal Case**.

Controller-Methoden

Alle Methoden versehen wir in Laravel mit der sogenannten **Camel-Case-Notation**. Das erste Zeichen wird kleingeschrieben.

Beispiel: `get()`, `getAll()`, `deleteSelected()`

Außerdem sollten im Controller für normale Create-Update-Delete-Operationen, kurz CRUD, folgende Methoden genutzt werden:

- > `index()`
- > `create()`
- > `store()`
- > `show()`
- > `edit()`
- > `update()`

► `destroy()`

Benennung von Datenbankelementen

Datenbanktabellen werden im Plural und kleingeschrieben. Worttrennungen kennzeichnen wir mit Unterstrichen – dies nennt man **Snake Case**. Die Bezeichnung sollte im Plural sein.

Beispiel: *users, user_pictures, awarded_certificates*

Tabellenspalten werden kleingeschrieben und nutzen ebenfalls Snake Case. Die Tabellenbezeichnung wird nicht erwähnt.

Beispiel: *id, created_at, updated_at, name, email*

Der **Primärschlüssel** sollte *id* heißen.

Fremdschlüssel enden auf *_id*. Beispiel: *user_id, post_id*

Variablen

Variablen werden im Singular mit **Camel Case** bezeichnet.

Beispiel: *\$activeUser*

Ausnahme: Sobald die Variable mehrere Items, ein Array oder eine Collection enthält, wird die Variable im Plural bezeichnet.

Models

Models sollten großgeschrieben im Singular bezeichnet werden und die Pascal-Case-Schreibweise nutzen.

Beispiel: *User, Role, UserImage*

Relationships

Wie bei allen Methoden wird die Camel-Case-Schreibweise verwendet.

Beispiel: *email(), postComment()*

Views

Views werden in der Snake-Case-Schreibweise und kleingeschrieben, außerdem schließen sie immer ab mit der Dateiendung *.blade.php*.

Beispiel: *register.blade.php, profile.blade.php*



Die verschiedenen Case Styles findest du hier: <https://medium.com/better-programming/string-case-styles-camel-pascal-snake-and-kebab-case-981407998841>

3.5 Teste dein Wissen!

1. Die View ...

Bitte ankreuzen:

repräsentiert die Benutzeroberfläche.

- repräsentiert die Schnittstelle zwischen Datenverwaltung und Benutzeroberfläche.
- repräsentiert das Request Handling.
- repräsentiert die Datenverwaltung.

2. Warum verwendet man die MVC-Architektur?

Bitte ankreuzen:

- Die Rollenverteilung im Team wird einfacher.
- Die MVC-Architektur verleiht dem Code mehr Struktur.
- Die Models und Views können separat voneinander bearbeitet werden, da diese nicht direkt voneinander abhängig sind.
- Das Prinzip der Separation of Concerns (auf Deutsch »Trennung der Zuständigkeiten«) wird umgesetzt.

3. Die Schreibweise `UserController` bezeichnet man als ...

Bitte ankreuzen:

- Pascal Case
- Snake Case
- Camel Case
- Kebab Case

4. Die Schreibweise `deleteSelectedFile()` bezeichnet man als ...

Bitte ankreuzen:

- Camel Case
- Snake Case
- Pascal Case
- Kebab Case

```
$request->session()->get('name', "Lorem Ipsum");
```

4. Prüfe, ob es in der Session einen Key '*name*' mit zugeordnetem Wert gibt.

```
session()->has('name');
```

Index

A

Abstracts 18
Accessors 138
Aggregatfunktionen 108 118
Anonyme Komponenten 77-78
Arrow-Funktionen 21
Attribute Bag 77-78

B

Blade Component Tags 77
Blade 26 35 63-78 149-155 183 192-193
Boilerplate 42
Brute-Force-Attacken 11

C

Callback-Funktion 38 44-46 63 145
Camel Case 26-27 177
Cascading 89
Case Styles 26
Chunking 108 112 118 188
Clean Code 16 152
Closures 21 44-46 54 58 105 137 181
Collections 118 144-146
Column Modifier 86 91 185
Containerisierung 28
Contracts 50-53
Controller Action 44-49 56 63 76 98 115 150-153 180
Controller Routes 39-40
Controller 23-25 35-50 56-66 70 76-77 93-94 98 114-115 150-153 177-182
Controller-Methoden 25 44
Cookies 164-166
Cross-Site-Scripting 67
CRUD Delete 119
CRUD Read 117

CRUD 25 44-49 93 114-120 148 180
CSRF Token 150 155-158 193-194
CSRF-Schutz 11
Custom Actions 48

D

DBAL 87
Debugging 96
Deklaration 47 95 187
Delete 25 38 46-49 89 93-94 107 119 149-150 180 193
Dependency Injection 44 50-56 84-85
Dependency Inversion: Details 16
Dependency Inversion 16 20 50
Deploy Application 168

E

Eager Loading 135-138 147
Eloquent Collections 146
Eloquent Model 115
Eloquent 81 93 98 114-146 156 169 189-191
Encrypter 157

F

Facades 50-53
Factories 35 159-162
Formulare 67 148-149 169
Fremdschlüssel 26 89 122-125 133 138

G

Generator 17 146 161-162
Github-Account 13

H

Has Many Through 126
Has One Through 125

Hashing 11 157-158 193
Helper 51-53 58-62 67 134 139 144-145 149 157-160 164-166 182 194
Homestead 28-32 38 81-82 90 121
HTTP-Methode 47-49 57 149-150 180
HTTP-Request 56
HTTP-Response 56
HTTP-Routing 37
HTTP-StatusCode 39 56-61 118

I

IDE 13
If-Abfragen 68-69
Include 74
Indexe 84 88 115
Inheritance 17-18 72 79 146 184
Initialisierung 47 93
Inline-PHP 69
Insert 93 97-98 106 111 187
Interface Injection 50
Interface Segregation Principle 16
Interfaces 20 52-53
Inversion of Control 50
IoC Container 50

J

Javascript 21 35 42 60 67 72-79 183-184
Joins 109
JSON 58-60 77 86 104-105 112 188

K

Kebab Case 27 77 177

L

Lazy Collections 118 146
Lazy Eager Loading 138
Life Cycle 24

Liskov Substitution Principle
16

M

Many to Many 124 131-134
142 190
Mass Assignment 116-120
133 141-142 160 189-191
Mass Update 116
Method Chaining 121 125
133-135 144-145
Method Injection 50 153
Migrations 35 42 83 87-92
98 121 186
Mutators 138
MVC-Architektur 12 23-27 63
177
MySQL 28 81 121

N

Nicknames 39-40

O

Objektrelationale Abbildung
114
One To Many 123 129-131
142 190
One To One 121 127-130 142
190
Open Closed Principle 16
ORM 114 140 189

P

Pascal Case 25-27 139 177
PDO-Daten 93
PHP Artisan 40-43 67 77 83
90-92 98 110 114 121 126
140 148-152 157-164 168
179 186-189 194

Pivot-Tabelle 124-125
131-134 142 191
Polymorphic Relationships 127
Props 77-78

Q

Query Chaining 100 110
Query Scope 120
Query-String 57 61

R

Raw Expressions 111
Raw Queries 93-96 156
React 35 42
Rekursion 20 169
Resource Controller 44-46 114
Routing 24 37-39 44-52 67
168 180

S

Schema Builder 83-89
Schleifen 69
Seeding 159
Select 93 103 119
Separation of Concerns 23 27
44 177
Service Container 50-53
Session 152 164-167 194-195
Single Action Controller 45-48
180
Single Responsibility Principle
16
Snake Case
SOLID 16 50
Spread Operator in Arrays 21
SQL Injection 156
SQL-Datenbanken 81
Stacks 75
Statement 93-94 105 109
stdClass 93-103 112 117
187-188
Subqueries 119

Switch-Abfragen 69

T

Template Engine 35 63
Template Inheritance 72 79
184
Tinker 126 143 160 191
Traits 18
Typed Properties 22

U

Unions 109
Update 25 46-49 93-94 106
115-117 149-152 180
URI 37-38 47 57 61 67
URL 24 37-41 56-60 81 169
179

V

Vagrant 21 28-32 36 90 178
Validierung 95 148-156 165
169 187 193
Verschlüsselung 156-157 164
Views 23-27 35-38 63-66 72
79 168 177 182-184
Vue 35 42 77

W

Weiterleitung 38-40 60-62 67
178 182
when-Abfragen 106
where-Abfragen 103-105
where-Closures 105

X

XSS-Angriffe 67 155-156