Laravel

Einleitung

Laravel ist ein freies PHP Web Framework, das das MVC-Pattern implementiert. Ein Framework ist ein überaus hilfreiches Code-Bundle, was einem viel Zeit und Nerven erspart. Im Prinzip handelt es sich hauptsächlich um die Bereitstellung von Funktionen, die theoretisch für jedes Projekt verwendet werden können.

Arbeitsumgebung einrichten

PHPStorm

Ich verwende für Laravel PHPStorm von JetBrains, natürlich kann man auch anderes verwenden, wie beispielsweise Visual Studio Code. In diesem Tutorial wird aber auf PHPStorm zurückgegriffen. Zudem wird XAMPP benötigt.

Dependency Manager Composer

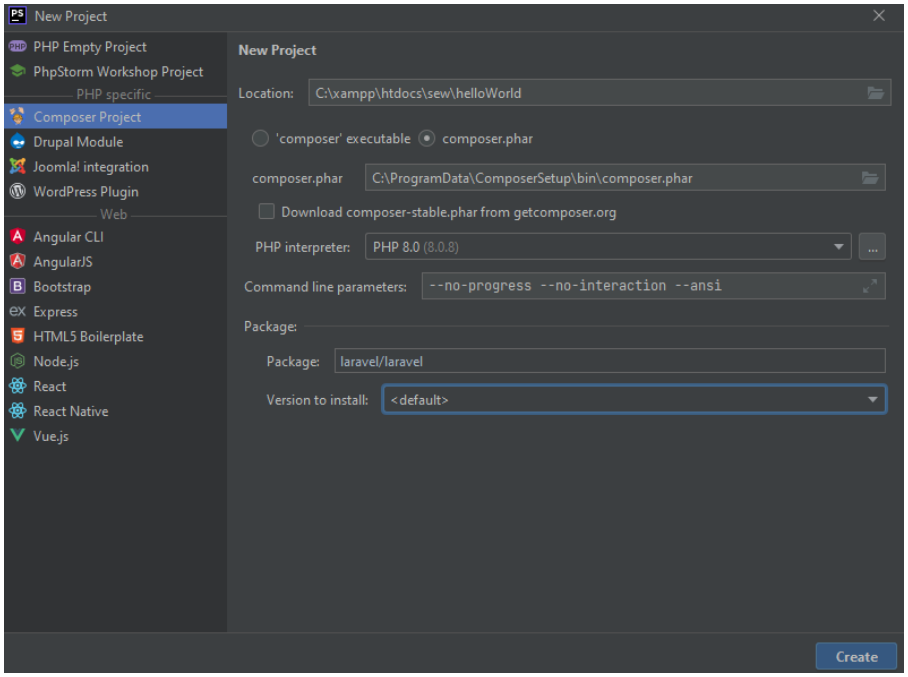
Composer ist ein sog. Dependency Manager für PHP, vergleichbar mit dem Node Package Manager (NPM) für Node.js. Composer erfordert PHP, entsprechende Informationen sind auf der Projekt-Website (https://getcomposer.org/doc/00-intro.md) zu finden.

Für die Installation wird einfach das Installations-Tool heruntergeladen (https://getcomposer.org/download/). Als nächstes muss die `PATH` Systemvariable geändert werden. In meinem Fall wäre das `C:\xampp\php`. Damit aber auch der Composer überall verwendet werden kann, muss dieser ebenfalls in die `PATH` Systemvariable hinzugefügt werden (`C:\ProgramData\ComposerSetup\bin`). Vermutlich muss danach neugestartet werden. Zur Überprüfung kann composer -version in der CMD eingegeben werden.

Übung

Erstellung eines Laravel-Projektes

Nun wird PHPStorm gestartet und ein neues Projekt erstellt. Dabei werden folgende Einstellungen getroffen:



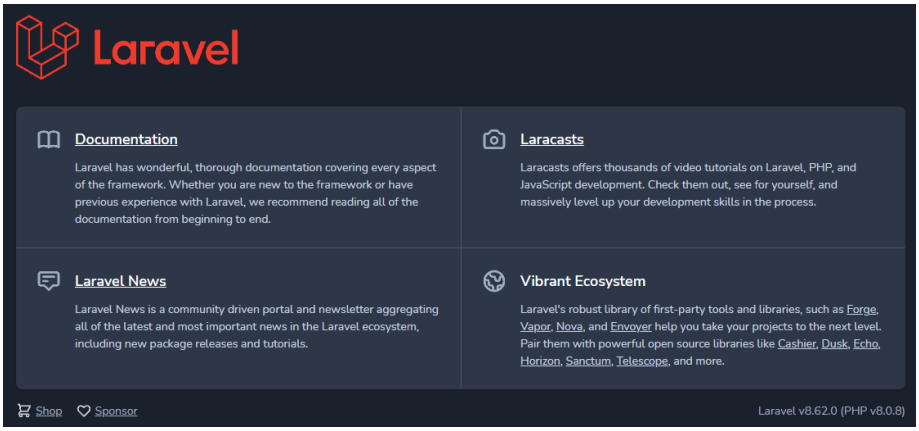
Achtung! Dabei muss Web-Server via XAMPP im Hintergrund laufen, ansonsten bekommt man einen Fehler.

Zweite Möglichkeit wäre das ganze über die CMD zu machen. Hierfür in der CMD einfach in den entsprechenden Ordner gehen und den folgenden Befehl ausgeben:

```
composer create-project --prefer-dist laravel/laravel helloWorld
```

Danach PHP Storm öffnen und ein neues Projekt Empty PHP Project erstellen
und hierbei das Verzeichnis `htdocs/sew/helloWorld` als Projekt-Root
setzen. Will man den Composer trotzdem in der IDE nutzen, ist dieser bekanntzumachen:
`Tools => Composer => Init Composer.`
Um die Integration zu testen, Tools => Run Command öffnen und die Versionsabfrage wiederholen.

Zum Testen der Applikation kann man nun den Pfad
`//127.0.0.1/sew/helloWorld/public/`



Einbindung der Datenbank

Hier für geht man in das `.env`-File im root-Verzeichnis und passt die relevanten Einstellungen an:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=sew
DB_USERNAME=sew
DB_PASSWORD=sew
```

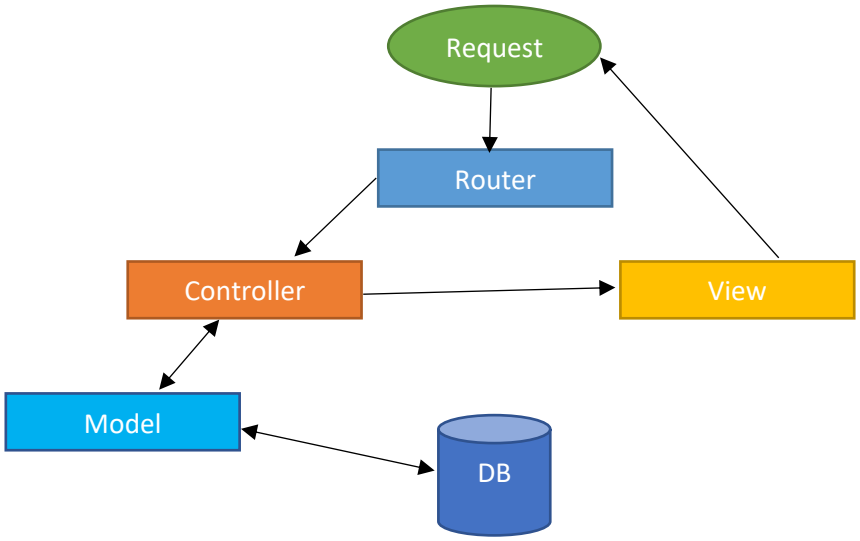
Hier wird wieder XAMPP für die Datenbank verwendet und der Benutzer `sew` bekommt alle Rechte. Zum Testen braucht es `Artisan` und die `Tinker`-Shell. Artisan ist ein Standard-Tool was mit Laravel mitgeliefert wird und hat nützliche Funktion, welche mit dem Befehl `php artisan list` eingesehen werden können.

Zum Testen wird einmal Tinker geöffnet (`php artisan tinker`) und danach der Befehl `DB::connection()->getPDO();` eingegeben. Unten sieht man ein mögliches Ergebnis:

```
PS C:\xampp\htdocs\sew\helloWorld> php artisan tinker
Psy Shell v0.10.8 (PHP 8.0.8 — cli) by Justin Hileman
>>> DB::connection()->getPDO()
=> PDO {#3492
     inTransaction: false,
     attributes: {
       CASE: NATURAL,
       ERRMODE: EXCEPTION,
       AUTOCOMMIT: 1,
       PERSISTENT: false,
       DRIVER_NAME: "mysql",
       SERVER_INFO: "Uptime: 1221  Threads: 6  Questions: 885  Slow queries: 0  Opens: 98  Flush tables: 1  Open tables: 91  Queries per second avg: 0.724",
       ORACLE_NULLS: NATURAL,
       CLIENT_VERSION: "mysqlnd 8.0.8",
       SERVER_VERSION: "5.5.5-10.4.20-MariaDB",
       STATEMENT_CLASS: [
         "PDOStatement",
       ],
       EMULATE_PREPARES: 0,
       CONNECTION_STATUS: "127.0.0.1 via TCP/IP",
       DEFAULT_FETCH_MODE: BOTH,
     },
   }
>>>
```

Zentrale Merkmale

Laravel basiert auf der MVC-Architektur. Folgende Abbildung zeigt einen vereinfachten Ablauf bei eingehendem Request.



MVC wird durch folgende Verzeichnisse und Dateien des erstellten Laravel-Projektes realisiert:

Ebene	Applikationspfad	Datei(en)
Model	app/	User.php
View	resources/views	welcome.blade.php
Controller	app/http/Controllers	Auth/AuthController.php Auth/PasswordController.php

Laravel würde theoretisch schon mit einem mitgelieferten Login-System kommen, deswegen auch diese speziellen Dateien in den Pfaden.

Routen

Jeder Request ist Laravel als Route bekanntzumachen. In der Regel geschieht das in der Datei `routes/web.php`. Hier findet man derzeit nur eine Route:

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

Diese leitet uns auf die `welcome.blade.php` einfach weiter. Um einfach Views zurückzugeben wird einfach die Methode `view` verwendet. Allerdings ist es empfehlenswerter (und üblicher) gleich auf Controller zu verweisen, wie es später auch gezeigt wird.

Views

Views verwenden ganz normales HTML/CSS, wobei CSS in Laravel extra behandelt wird. Die Views können in resources/views gefunden werden. In den Routes kann dann einfach mit dem Attribut `view(´welcome´)` auf eine beliebige View verwiesen werden, allerdings, wie bereits erwähnt, ist es gewöhnlicher auf einen Controller zu verweisen.

Blade

Die Views werden immer `<name>.blade.php` genannt. Blade ist ein mächtiges Template was vieles vereinfacht. Beispielsweise sind Schleifen, if-Abfragen und mehr verfügbar. Die ganzen „Funktionen“ können mit `@` abgerufen werden. Mit @yield kann HTML-Code eingebunden werden, welcher vorher in einer @section definiert wurde.

Beispiel:

@yield kann überall (meist im body) eingebunden werden. Der Inhalt wird mit einem Namen erstellt, hier ist es „content.“ (Name dieser File: `app.blade.php`)

```
<!doctype html>
<html lang="en">
	<head>
		<meta charset="UTF-8">
		<meta name="viewport"
		content="width=device-width, user-scalable=no, initial-
		scale=1.0,
		maximum-scale=1.0, minimum-scale=1.0">
		<meta http-equiv="X-UA-Compatible" content="ie=edge">
		<title>Dashboard</title>
	</head>
	<body>
		<ul>
		<li><a href="/">Dashboard</a></li>
		<li><a href="projects">Projects</a></li>
		<li><a href="customers">Customerss</a></li>
		</ul>
			@yield('content') <!—View-spezifischer Inhalt! -->

	</body>
</html>
```

Erstellt man nun eine neue view.blade.php, so kann man mit @extends den eigentlichen File „erweitern“. Bespiel: (Name dieser File: `projects.blade.php`)

```
@extends('layouts/app') <!-- Bezug zum Template herstellen -->

@section('content') <!-- Inhalte für Section content definieren -->

	<h1>Projects</h1>

@endsection
```

Wird die obige View (`projects.blade.php`) aufgerufen, so wird das gesamte Template von `app.blade.php` geladen, allerdings wird `@yield` ausgetauscht und durch den `@section(´content´).`

Controllers

Grundsätzlich können in Laravel auch Controller zum Einsatz kommen. Diese sind einfacher in der Verwendung, beziehungsweise vereinfacht, dass nicht tausende von Routes geschrieben werden müssen.

Erstellt werden Controller mithilfe von Artisan. Hierfür einfach `php artisan make:controller` eingeben. Controller werden einfach in der Route angegeben und dann können die jeweiligen Methoden, die im Controller definiert wurden, aufgerufen werden.

Route mit Controller:

```
Route::get('/', 'PagesController@dashboard');
```

Wie bereits erwähnt können in einem Controller beliebig viele Methoden erstellt werden. Beispielsweise können so eine View angezeigt und an eine View gesendet werden:

```
class PagesController extends Controller
{
	public function dashboard(){
		$tasks = [
			'INSY lernen.',
			'SYTI-HÜ erledigen.',
			'ITP-Projekte abschließen.'
		];

		return view('dashboard', ['tasks' => $tasks]);
	}
}
```

Im PagesController wird die Funktion dashboard erstellt, welche ein Array beinhaltet. Dieser Array wird dann in der markierten Zeile an die dashboard-View weitergegeben und die gesamte View wird returniert.

Diese können in der dashboard.blade.php aufgerufen werden:

```
@section('content')
	<h1>Dashboard</h1>
	<ul>
		@foreach ($tasks as $task)
			<li> {{ $task }} </li>
		@endforeach
	</ul>
@endsection
```

Natürlich kann nicht nur ein Array, sondern auch eine Datenbank abfrage getätig werden und diese dann übermittelt werden. Laravel bedient sich auch Migrationen, also kann man selbst die Entscheidung treffen, ob man direkt auf die Datenbank zugreift oder Model-Klassen verwendet.

Laravel erstellt bereits bei der Installation zwei Migrations (siehe Verzeichnis `database/migrations`), die die Grundlage, für die von Laravel bereitgestellte Authentifizierung-Funktionalität bilden.

In der Migartion gibt es die Funktion up, welche einfach eine Tabelle mit allem drum und dran erstellt. Beispielsweise die User-Tabelle, welche automatisch mitgeliefert wird:

```
public function up()
{
	Schema::create('users', function (Blueprint $table) {
		$table->bigIncrements('id');
		$table->string('name');
		$table->string('email')->unique();
		$table->timestamp('email_verified_at')->nullable();
		$table->string('password');
		$table->rememberToken();
		$table->timestamps();
	});
}
```

Diese Methode wird ausgeführt, wenn man den Befehl `php artisan migrate` ausführt. Eine Migration kann erstellt werden mit dem Befehl. Beispielsweise:

```
php artisan make:migration create_customers_table
```

Die Tabelle wird dann `customers` heißen, dies erkennt Laravel automatisch.

Models

Ein Customer-Model können mit folgendem Befehl erstellt werden.

```
php artisan make:model Customer
```

Models können ebenfalls für Datenabfragen verwendet werden:

```
public function customers(){

	$customers = \App\Customer::all();

	return view('customers', ['customers' => $customers]);
}
```

Nun wären die Daten die in $customers abgefragt werden, in der Customer-View verfügbar.