Laravel Datenbankzugriff Tutorial (CRUD)

Quelle

http://www.expertsphp.com/laravel-6-crud-create-read-update-delete-generator-for-beginners-with-example/

Fertiges Projekt: https://github.com/gronowski/LaravelCRUD

Allgemeines

CRUD bedeutet Create, Read, Update, Delete, also die typischen Datenbank-Operationen.

Zurzeit ist Laravel 6.0 aktuell (6.9.19). Testen mit *php artisan -V.*
PHP muss mindestens Version 7.2 sein, sonst funktioniert Laravel 6.0 nicht.

Das erste Ziel ist, mithilfe eines Forms in der Seite *Blogs* eine Datenbankzeile in eine Mysql-Tabelle zu speichern und diese darzustellen und zu editieren.

Die kopierten Code Snippets sollten in Visual Studio Code mit einem Beautifier/Formatter formatiert werden. Dies ist mit Ctlrl A (alles markieren) und mit Shift Alt f (Formatieren) möglich. Allenfalls muss der Beautifier/Formatter noch installiert werden. Falls der Formatter noch nicht installiert ist, erscheint ein Menu unten rechts.

Die Ports 80 und 8000 sollten nicht schon durch andere Anwendungen belegt sein. Vor allem bei Port 80 stören Installationen von Skype und Internet Information Server IIS (Webserver von Microsoft).

Laravel teilt, wie die meisten Frameworks, den Code in Model - View - Controller auf. Dies führt zu einer besseren Wartbarkeit, weil sich die einzelnen Komponenten separat editieren lassen. Andererseits führt es zu mehr und abstrakterem Code.

Tutorials zu Laravel sollten sich auf die Version 6.0 oder höher beziehen, weil mit der Version 6.0 diverse Änderungen vollzogen wurden. Unter techiediaries.com gibt es zahlreiche weitere Tutorials, z.B.eine gute Einführung auf
https://www.techiediaries.com/laravel-tutorial/

Laravel installieren

Am besten ist es, wenn man Laravel in ein Unterverzeichnis von Xampp installiert (c:\xampp\htdocs\laravel. Xammp wird benötigt um mit Phpmyadmin auf die Datenbank zugreifen zu können.

1. Composer per Windows Installer
2. PHP und MySQL per XAMPP (wichtig neueste PHP Version, Version > 7.2)
3. Node.js per Windows Installer
4. Laravel 6.0 per Kommandozeile installieren (composer global require laravel/installer)

Der Fehler, dass das ***vendor*** Verzeichnis nicht erzeugt wird, ist durch eine zu alte PHP Version verursacht.. Wird ein PHP der Version <7.2 verwendet, tritt dieser Fehler auf. Vor der Installation deshalb genau beachten, welche Version von XAMPP installiert ist.

Server.php soll mit ***IDP.generic*** Virus befallen sein (AVAST). Ist eine generische Warnung und könnte keine Bedrohung darstellen.

Projekt erzeugen

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

oder

```
laravel new projektname (laravel new blog)
```

=> Falls keine vendor Verzeichnis erzeugt wird, ist die PHP-Version zu alt <7.2.

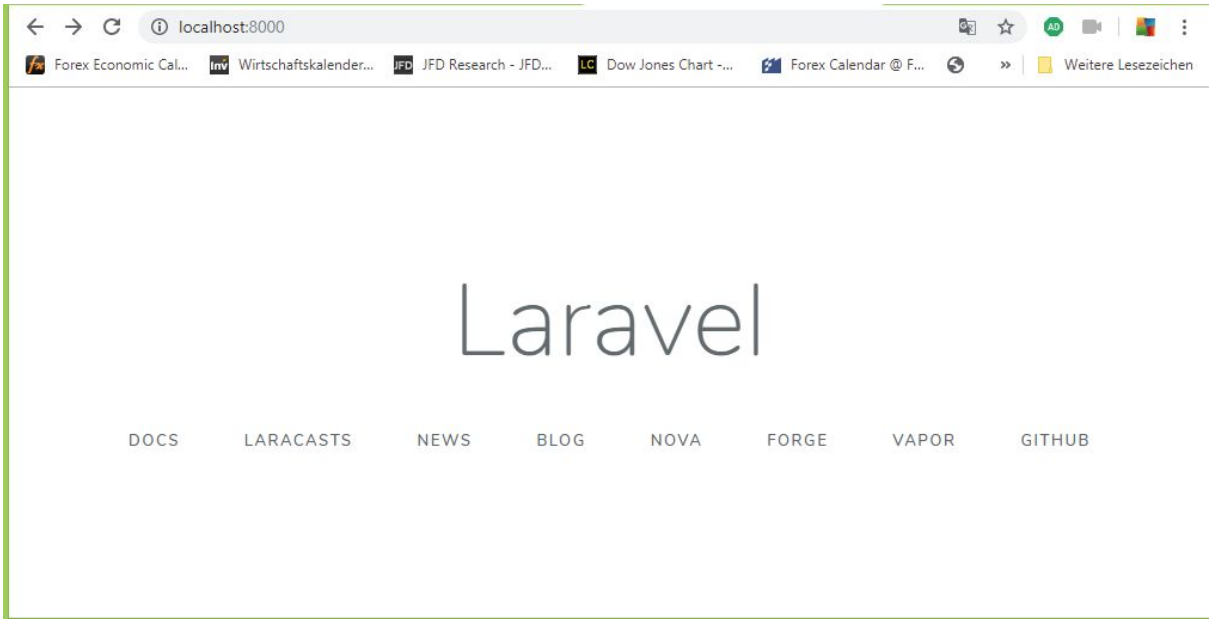
Die Version von Laravel abrufen: `php artisan -V`

Lokaler dev Server starten

`php artisan serve` (im Projektverzeichnis)

Seite (Homepage) per angegebener Url aufrufen (http://localhost:8000/)

Probleme: Avast Virenscanner gibt Warnung, dass ein Trojaner vorhanden sein könnte (***IDP.generic).***



MySQL muss ebenfalls gestartet sein. Auch Apache, falls man mit phpmyadmin auf die Datenbank zugreifen will. Am besten erfolgt dies über XAMPP.

Datenbank konfigurieren (.env Datei bearbeiten)

Normalerweise sollte die .env Datei schon existieren und für eine Mysql-Datenbank die korrekten Werte enthalten. Je nach dem wie das Projekt installiert wurde, kann die Datei auch .envExample heissen und muss auf .env umbenannt werden.

Für Übungszwecke ist der MySQL-User = root und das Passwort leer.

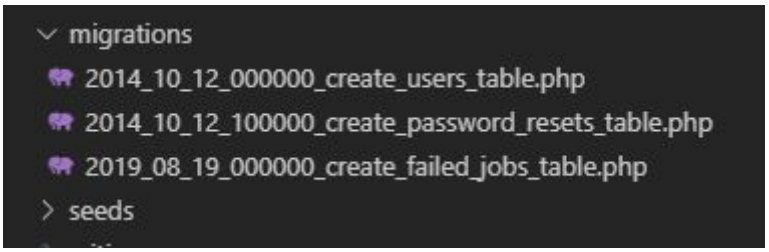
```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Datenbank und Tabelle anlegen

Mit MySQL PhpMyAdmin DB anlegen, zum Beispiel mit Namen *laravel*

Model: Tabelle anlegen, alte Migrationsfiles löschen

Unter `database>migrations` alte Migration-Files löschen.



Dann Konsolenbefehl:

```
php artisan make:migration create_blogs_table --create=blogs
```

Resultat:

Es wird eine Migrationsdatei erzeugt, die editiert werden muss. Migrationsfile 2019_09_05_184257_creade_blogs_table.php editieren und Db-Spalten einfügen

```
public function up()
    {
        Schema::create('cruds', function (Blueprint $table) {
            $table->bigIncrements('id');

            $table->string('blog_title');
            $table->text('blog_content');

            $table->timestamps();
        });
    }
```

Nach dem Editieren:

Konsole: `php artisan migrate`

‘blogs’ und ‘migrations’ Tabellen werden in Mysql angelegt.

Es können unter Umständen Fehlermeldungen auftauchen, z.B. SQL-Fehler.

Mit phpmyadmin überprüfen Sie, ob eine leere Tabelle erzeugt wurde.

Routes

In routes/web.php die Zeile `Route::resource('blogs','blogController');`

unterhalb der Route für die Homepage einfügen. Neben der Route auf die Homepage wird nur diese eine Route benötigt, obwohl auf mehrere Seiten zugegriffen wird.

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

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

Route::resource('blogs','blogController');
```

Die statische Methode Route::resource() bildet mehrere Pfade ab um diverse Aktionen mit der Resource durchzuführen.

- GET `/contacts`, mapped to the `index()` method,
- GET `/contacts/create`, mapped to the `create()` method,
- POST `/contacts`, mapped to the `store()` method,
- GET `/contacts/{contact}`, mapped to the `show()` method,
- GET `/contacts/{contact}/edit`, mapped to the `edit()` method,
- PUT/PATCH `/contacts/{contact}`, mapped to the `update()` method,
- DELETE `/contacts/{contact}`, mapped to the `destroy()` method.

Das Model

Beim Model-View-Controller Pattern stellt das Model die Daten dar.

Das Model mit dem Befehl

```
php artisan make:model Blog
```

erzeugen.

app\Blog.php ist das Model, d.h. die Datenstruktur

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Contacts extends Model
{
    //
}
```

Die Datei muss von Hand ergänzt werden (die schliessende PHP Klammer fehlt):

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Blog extends Model
{
protected $fillable = [
      'blog_title',
      'blog_content'
      ];
}
```

Diese Einträge führen zu folgender Tabellenstruktur:

id, created_at und updated_at müssen im Model nicht angegeben werden. Sie wurden bei der Erzeugung der Tabelle angegeben.

Controller erzeugen

Im MVC-Pattern stellt der Controller die Datenbankmanipulationen dar.

`Konsole: php artisan make:controller blogController --resource`

Controller created successfully =>

`app/Http/Controllers/blogController.php`

wird erzeugt. Darin ist das Gerüst aller Kontroller-Methoden vorhanden, die zum Model (Datenstruktur) passen. Die Codestellen mit den fett gedruckten “//” müssen ergänzt werden.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ContactController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
```

```
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
```

```
    {
        //
    }
}
```

Implementierung Methoden in blogController.php

Der blogController beinhaltet sieben Methoden, die wir in diesem Schritt ergänzen. Zudem muss das Model App\Blog in die Datei eingeschlossen werden.

```
<?php
namespace App\Http\Controllers;
use App\Blog;
use Illuminate\Http\Request;
class blogController extends Controller
{

/**
* Display a listing of the resource.
*
* @return \Illuminate\Http\Response
*/
public function index()
{
     $blogs = Blog::orderby('id', 'desc')->get();
     return view('blogs.index',compact('blogs'));
}


/**
* Show the form for creating a new resource.
*
* @return \Illuminate\Http\Response
*/
public function create()
{
     return view('blogs.create');
}


/**
* Store a newly created resource in storage.
*
* @param \Illuminate\Http\Request $request
```

```
* @return \Illuminate\Http\Response
*/
public function store(Request $request)
{
     $request->validate([
     'blog_title' => 'required',
     'blog_content' => 'required',
     ]);
     Blog::create($request->all());
     return redirect()->route('blogs.index')
     ->with('success','blogs created successfully.');
}

/**
* Display the specified resource.
*
* @param \App\Blog $blog
* @return \Illuminate\Http\Response
*/
public function show(Blog $blog)
{
     return view('blogs.show',compact('blog'));
}

/**

* Show the form for editing the specified resource.
*
* @param \App\Blog $blog
* @return \Illuminate\Http\Response
*/
public function edit(Blog $blog)
{
     return view('blogs.edit',compact('blog'));
}

/**
* Update the specified resource in storage.
*
* @param \Illuminate\Http\Request $request
* @param \App\Blog $blog
* @return \Illuminate\Http\Response
*/
public function update(Request $request, Blog $blog)
{
     $request->validate([
```

```
        'blog_title' => 'required',
        'blog_content' => 'required',
        ]);
        $blog->update($request->all());
        return redirect()->route('blogs.index')
        ->with('success','blogs updated successfully');
}


/**
* Remove the specified resource from storage.
*
* @param \App\Blog $blog
* @return \Illuminate\Http\Response
*/
public function destroy(Blog $blog)
{
        $blog->delete();
        return redirect()->route('blogs.index')
        ->with('success','Blog deleted successfully');
}
}//Ende Class
```

Die Bedeutung der einzelnen Methoden ist im Kommentar gegeben, der mit /** startet..

View (Blade Files)

Die View umfasst alle Darstellungen in html und css. Blade ist ein Rendering-Tool.

Erzeugen Sie einen Unterordner blogs im views Ordner. Dann erzeugen Sie folgende Dateien:

1. layout.blade.php (**resources/views/blogs/layout.blade.php**)

2. index.blade.php (**resources/views/blogs/index.blade.php**)

3. create.blade.php (**resources/views/blogs/create.blade.php**)

4. edit.blade.php (**resources/views/blogs/edit.blade.php**)

5. show.blade.php (**resources/views/blogs/show.blade.php**)

Kopieren sie die folgenden Codes in die Dateien:

resources/views/blogs/layout.blade.php

```
<!DOCTYPE html>
<html>

<head>
    <title>Laravel 6.0 CRUD Generator Application</title>
    <link
href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0
.0-alpha/css/bootstrap.css" rel="stylesheet">
</head>

<body>
    <div class="container">
        @yield('content')
    </div>
</body>

</html>
```

Grund-Layout, das von anderen Layouts extended wird.

resources/views/blogs/index.blade.php

```
@extends('blogs.layout')
@section('content')
<div class="row">
     <div class="col-lg-12 margin-tb">
          <div class="pull-left">
               <h2>Laravel 6.0 CRUD Example</h2>
          </div>
          <div class="pull-right">
          <a class="btn btn-success" href="{{
     route('blogs.create') }}"> Create New Blog</a>
          </div>
     </div>
</div>
@if ($message = Session::get('success'))
<div class="alert alert-success">
     <p>{{ $message }}</p>
```

```
</div>
@endif
<table class="table table-bordered">
     <tr>
          <th>Blog Title</th>
          <th>Blog Content</th>
          <th width="280px">Action</th>
     </tr>
     @foreach ($blogs as $blog)
     <tr>
          <td>{{ $blog->blog_title }}</td>
          <td>{{ $blog->blog_content }}</td>
          <td>
               <form action="{{ route('blogs.destroy',$blog->id)
          }}" method="POST">
               <a class="btn btn-info" href="{{
          route('blogs.show',$blog->id) }}">Show</a>
               <a class="btn btn-primary" href="{{
          route('blogs.edit',$blog->id) }}">Edit</a>
          @csrf
          @method('DELETE')
          <button type="submit" class="btn
     btn-danger">Delete</button>
               </form>
          </td>
     </tr>
     @endforeach
</table>
@endsection
```

resources/views/blogs/create.blade.php

```
@extends('blogs.layout')
@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Add New Blogs</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{
route('blogs.index') }}"> Back</a>
        </div>
    </div>
</div>
@if ($errors->any())
```

```
<div class="alert alert-danger">
    <strong>Whoops!</strong> There were some problems with your
input.<br><br>
    <ul>
        @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
@endif
<form action="{{ route('blogs.store') }}" method="POST">
    @csrf
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Blog Title:</strong>
                <input type="text" name="blog_title"
class="form-control" placeholder="title">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Blog Content:</strong>
                <textarea class="form-control" name="blog_content"
placeholder="Content"></textarea>
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn
btn-primary">Submit</button>
        </div>
    </div>
</form>
@endsection
```

resources/views/blogs/edit.blade.php

```
@extends('blogs.layout')
@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Edit Blogs</h2>
        </div>
        <div class="pull-right">
```

```
            <a class="btn btn-primary" href="{{
route('blogs.index') }}"> Back</a>
        </div>
    </div>
</div>
@if ($errors->any())
<div class="alert alert-danger">
    <strong>Whoops!</strong> There were some problems with your
input.<br><br>
    <ul>
        @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
@endif
<form action="{{ route('blogs.update',$blog->id) }}"
method="POST">
    @csrf
    @method('PUT')
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Blog Title:</strong>
                <input type="text" name="blog_title" value="{{
$blog->blog_title }}" class="form-control" placeholder="Name">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Blog Content:</strong>
                <textarea class="form-control" name="blog_content"
placeholder="Detail">{{ $blog->blog_content }}</textarea>
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn
btn-primary">Submit</button>
        </div>
    </div>
</form>
@endsection
```

resources/views/blogs/show.blade.php

```
@extends('blogs.layout')
@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2> Show Blogs</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{
route('blogs.index') }}"> Back</a>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Blog Title:</strong>
            {{ $blog->blog_title }}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Blog Content:</strong>
            {{ $blog->blog_content }}
        </div>
    </div>
</div>
@endsection
```

Jetzt kann der lokale Server mit

```
php artisan serve
```

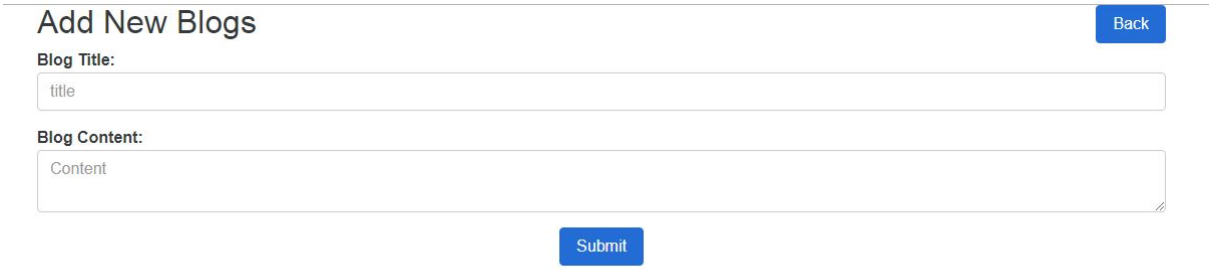
im Projektverzeichnis gestartet werden.

http://localhost:800 zeigt die Homepage

http://localhost:8000/blogs

Stellt das Formular dar.

neuer Blog hinzu fügen



Aufgaben

Betrachten Sie die einzelnen Seiten in der Html-Ansicht des Browsers. Betrachten Sie vor allem die Pfade in den Form Tags.

Erstellen Sie selbst schrittweise eine CRUD Applikation.