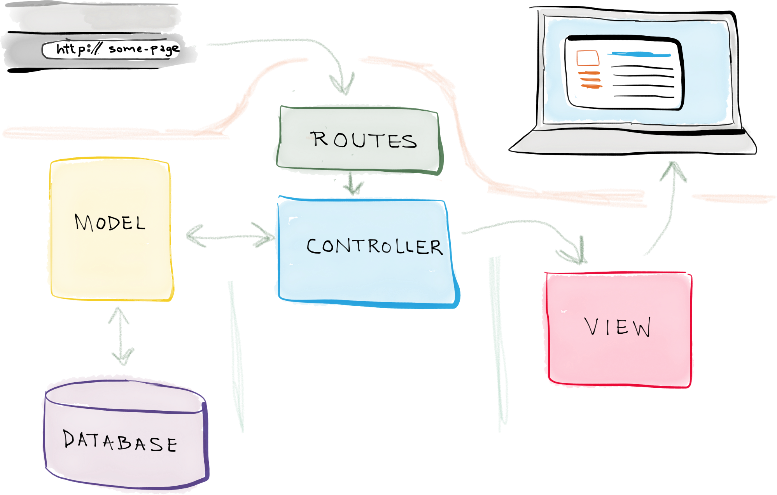
# Laravel

Laravel ist ein MVC Framework. MVC steht hierbei für Model View Controller und spiegelt eines der Grundkonzepte des Frameworks wider.

Das Herzstück im MVC-Pattern bietet das **Model**. Ein Model repräsentiert immer eine Einheit. Als Beispiel lässt sich hier sehr gut ein Nutzer Ihrer Applikation sehen. Somit ist ein Model ein Objekt, welches meist per ORM (Object-Relational Mapping) mit einer Datenbank verknüpft wird. In einer relationalen Datenbank wie zum Beispiel MySQL repräsentiert ein Model daher immer einen Datensatz.

Verknüpft mit dem Model sind auch alle Aktionen, die sich auf dieses Model anwenden lassen. Soll ein Nutzer angelegt und ihm ein Name bzw. eine Mail-Adresse zugewiesen werden, könnte dies in Laravel so aussehen:

Quelle: https://selftaughtcoders.com

$user = **new** User();  
$user->**name** = **'abraham'**;  
$user->**email** = **'support@sz-ybbs.ac.at'**;  
$user->save();

In Laravel werden Models standardmäßig direkt im app-Verzeichnis gespeichert.

Eine **View**, oder auch Template genannt, ist nichts anderes als die Schnittstelle zum Endanwender. Im Falle einer Website oder Web-Applikation ist dies meist HTML, welches das User Interface definiert. Um die Ausgabe effektiver zu gestalten und das HTML leserlicher darzustellen, hat Laravel viel Mühe in eine Templating-Sprache Namens Blade gesteckt. Ein Beispiel zum Ausgeben aller Nutzer und deren E-Mail-Adresse sieht darin wie folgt aus:

<**ul**>  
 **@foreach(**$users **as** $user**)** <**li**><**b>**{{ $user->**name** }}:</b> {{ $user->**email** }}</**li**>  
 **@endforeach**</**ul**>

In Laravel wird mittels **Routen** üblicherweise jede Seite und jede Aktion einer Seite einer bestimmten Methode in einem bestimmten **Controller** zugeordnet. Nehmen wir an, die oben genannte View soll unter der URL http://laravel.local/nutzer erscheinen. Dazu ist eine Klasse mit dem Namen UsersController zu erstellen, in der das Management der Benutzer stattfindet. Dort würde nicht nur das Auflisten stattfinden, sondern auch das Bearbeiten, Löschen oder Erstellen. Im nachfolgenden Beispiel könnte die URL /nutzer mit der index()-Methode des UsersController verbunden worden sein. Damit werden folgende Aktionen ausgeführt, sobald ein Besucher die URL aufruft:

**class** UsersController **extends** Controller  
{  
 **public function** index()  
 {  
 $users = User::*all*(); //Abfragen aller Datensätze aus dem Model  
 **return** view(**'userslist'**)->with(**'users'**, $users); //View aufrufen und   
 //Datensätze übergeben  
 }  
}

# Installation

Mit „Composer“ (siehe: <https://getcomposer.org/download/> ) den Laravel-Installer installieren:

cmd:

composer global require laravel/installer

Mit “Laravel” ein neues Projekt erzeugen:

In cmd in das Verzeichnis c:\xampp\htdocs\laravel wechseln und dort eine neue Installation starten:

laravel new wine

Erster Aufruf der Laravel Tutorial Demo Seite:

cd wine

php artisan serve

Jetzt kann die Demoseite unter <http://localhost:8000> aufgerufen werden.

# DATENBANK-Konfiguration

MariaDB-User laravel mit allen Rechten auf DB laravel anlegen:

CREATE USER 'laravel'@'localhost' IDENTIFIED BY 'htl';

CREATE DATABASE IF NOT EXISTS laravel default character set=utf8;

GRANT ALL PRIVILEGES ON laravel.\* TO 'laravel'@'localhost';

Datei “.env” mit den DB\_CONNECTION-Daten anpassen.

Datenbank-Migrations:

* PHP-Skripts zum “Neuanlegen von Datenbanktabellen, zum Erweitern der Datenbank, unabhängig von der Datenbank“
* Pro Tabelle typisch ein Skript
* Tabelle migrations protokollierte jede Migration

Model & Migration für Tabelle Winzer erzeugen:

c:\xampp\htdocs\laravel\wine

php artisan make:model Winemaker -m

(Nur Migration erzeugen: php artisan make:migration create\_winemakers\_table)

Migration-File:

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::create('winemakers', function (Blueprint $table) {

            $table->increments('id');

            $table->string('name');

            $table->string('street')->nullable();

            $table->integer('zipcode')->nullable();

            $table->string('city')->nullable();

            $table->string('phone')->nullable();

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('winemakers');

    }

};

Eloquent Models:

* Eloquent ist der Object Relational Manager (ORM) von Laravel
* Eine Model Klasse entspricht einer Tabelle
* Models *können* die wichtigsten Datenbankzugriffe automatisch.
* Models *kennen* alle Attribute der Datenbank.
* Attribute müssen im Model nicht codiert werden (Magic methods)

**class** Winemaker **extends** Model  
{  
 *//*}

Konventionen:  [https://laravel.com/docs/5.6/eloquent#eloquent-model-conventions](https://laravel.com/docs/5.6/eloquent)

* + Model: Großschreibung, CamelCase, Einzahl
  + Table: Kleinschreibung, Snake Case, Mehrzahl
  + Winemaker -> winemakers
  + SuperUser -> super\_users
  + Person -> people
  + Fish -> fish

Migrations einspielen:

* **php artisan migrate** – alle Migrations laufen (up())
* php artisan migrate:reset – alle Migrations rückwärts (down())
* php artisan migrate:rollback – letzter Batch down()
* php artisan migrate:refresh – down()-up()

Database Seeder:

* Klassen zum Befüllen von Datenbanktabellen (Grunddaten)
* Hier: Ausprobieren verschiedener Datenbankfunktionen
* php artisan make:seeder WineMakerSeeder

<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;

use Illuminate\Database\Seeder;

use Illuminate\Support\Facades\DB;

class WineMakerSeeder extends Seeder

{

    /\*\*

     \* Run the database seeds.

     \*/

    public function run(): void

    {

        DB::table('winemakers')->delete();

        DB::table('winemakers')->insert([

            'name' => 'Lackner Tinnacher',

            'street' => 'Steinback 8',

            'zipcode' => 4567,

            'city' => 'Gamlitz',

            'phone' => '1234567'

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Weingut Prager',

            'street' => 'Weissenkirchen 48',

            'zipcode' => 3610,

            'city' => 'Weissenkirchen',

            'phone' => '1234567'

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Weingut Emmerich Knoll',

            'street' => 'Unterloiben 10',

            'zipcode' => 3601,

            'city' => 'Unterloiben',

            'phone' => '1234456'

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Weingut F.X.Pichler',

            'street' => 'Unterloiben 27',

            'zipcode' => 3601,

            'city' => 'Unterloiben',

            'phone' => '11122233'

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Weingut Spätlese',

            'street' => 'Weintalstraße 23',

            'zipcode' => 1136,

            'city' => 'Wien',

            'phone' => null

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Freie Weingärten Wachau',

            'street' => 'Kremstalstraße 23',

            'zipcode' => 3600,

            'city' => 'Krems',

            'phone' => '2304002'

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Stiftskellerei',

            'street' => null,

            'zipcode' => null,

            'city' => null,

            'phone' => null

        ]);

        DB::table('winemakers')->insert([

            'name' => 'Weingut Biegler',

            'street' => 'Wienerstraße 88',

            'zipcode' => 2352,

            'city' => 'Gumpoldskirchen',

            'phone' => '54564565'

        ]);

        //

    }

}

Aufruf:

* php artisan db:seed --class=WineMakerSeeder

# Routen-Konfiguration

Die Konfiguration erfolgt in der Datei routes/web.php (falls Routenänderungen nicht übernommen werden, dann php artisan route:clear ausführen).

**Basic – Routing**

Aufruf einer Funktion (Closure):

Route::get('/', function () {

    return view('welcome');

});

Route::get('hello1', function () {

    return 'Hello World!';

});

Route::get('hello2/{abc}', function ($abc) {

    return 'Hello '.$abc.'!';

});

Route::get('hello3/{abc?}', function ($abc='Hammer') {

    return 'Hello '.$abc.'!';

});

Ein GET-Request auf  [http://127.0.0.1:8000/hello1](http://laravel.local/hello1) gibt “Hello World!” aus

Ein GET-Request auf  [http://127.0.0.1:8000/hello2/Brachinger](http://laravel.local/hello2/Brachinger) gibt “Hello Brachinger!” aus

Ein GET-Request auf  [http://127.0.0.1:8000/hello2](http://laravel.local/hello2) gibt einen Fehler aus

Ein GET-Request auf  [http://127.0.0.1:8000/hello3/Brachinger](%20http://127.0.0.1:8000/hello3/Brachinger) gibt “Hello Brachinger!” aus

Ein GET-Request auf <http://127.0.0.1:8000/hello3/> gibt “Hello Hammer!” aus

Die Funktion kann auch PHP-Code beinhalten, z.B.:

Route::get('dbtest', function () {

    if (DB::connection()->getDatabaseName()) {

        echo 'Connected successfully to db ' . DB::connection()->getDatabaseName();

    }

});

oder z.B. mittels all()-Aufruf alle Tupel der Relation ausgeben:

Route::get('winemaker1', function () {

    $winemakers=\App\Models\Winemaker::all();

    echo '<pre>';

    print\_r($winemakers);

    echo '</pre>';

 });

**Routing mit Controller - mit GET-Methode**

php artisan make:controller WinemakerController --resource

erzeugt die Datei app\Http\Controllers\WinemakerController.php

1. Aufruf der Methode index2 Methode der WinemakerController Klasse in der *callable array syntax* (verpflichtend ab Laravel 8):

Route::get('winemaker2', [App\Http\Controllers\WinemakerController::class, 'index2']);

Ruft die Methode index2 der Controllerklasse WinemakerController auf:

class WinemakerController extends Controller

{

    public function index2()

    {

        $winemakers=\App\Models\Winemaker::all();

        echo '<pre>';

        print\_r($winemakers);

        echo '</pre>';

    }

}

b) Ausgabe mittels View – PHP-Syntax

Route::get('winemaker3', [App\Http\Controllers\WinemakerController::class, 'index3']);

mit

    public function index3()

    {

        $winemakers=\App\Models\Winemaker::all();

        return view('winemakerlistsimple3', ['winemakers'=>$winemakers]);

    }

und zugehöriger View: (Datei winemakerlistsimple3.blade.php im Ordner resources/views anlegen)

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Winzer</title>

</head>

<body>

<table>

    <?php foreach ($winemakers as $winemaker) : ?>

    <tr>

        <td><?php echo $winemaker->id; ?></td>

        <td><?php echo $winemaker->name; ?></td>

        <td><?php echo $winemaker->street; ?></td>

        <td><?php echo $winemaker->zipcode; ?> <?php echo $winemaker->city; ?></td>

        <td><?php echo $winemaker->phone; ?></td>

    </tr>

    <?php endforeach; ?>

</table>

</body>

</html>

c) Ausgabe mittels View – BLADE-Syntax

Route::get('winemaker4', [App\Http\Controllers\WinemakerController::class, 'index4']);

mit

    public function index4()

    {

        $winemakers=\App\Models\Winemaker::all();

        return view('winemakerlistsimple4', ['winemakers'=>$winemakers]);

    }

und zugehöriger View: (Datei winemakerlistsimple4.blade.php im Ordner resources/views anlegen)

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Winzer</title>

</head>

<body>

<table>

    @foreach ($winemakers as $winemaker)

        <tr>

            <td>{{ $winemaker->id }}</td>

            <td>{{ $winemaker->name }}</td>

            <td>{{ $winemaker->street }} </td>

            <td>{{ $winemaker->zipcode }} {{ $winemaker->city }}</td>

            <td>{{ $winemaker->phone }}</td>

        </tr>

    @endforeach

</table>

</body>

</html>

Anmerkungen:

* Blade View Engine
  + HTML Templates mit Blade Content
  + File: viewname.blade.php
  + View: viewname
* übersetzt xyz.blade.php -> .php und speichert das Produkt im Cache ab.
* Gecachte View wird ab dann verwendet
* {{ $variable }} -> echo $variable + htmlspecialchars()
* HTML Sonderzeichen werden escaped und gefahrlos dargestellt
* {!! $variable !!} -> echo $variable – alle Zeichen werden „as is“ dargestellt.

CSS-Formatierung mittels Bootstrap:

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <title>Winzer</title>

</head>

<body>

<div class="container">

<h2>Winzer</h2>

<table class="table table-striped table-bordered">

    <thead>

    <tr>

        <th>ID</th>

        <th>Name</th>

        <th colspan="2">Adresse</th>

        <th>Telefon</th>

    </tr>

    </thead>

    <tbody>

    @foreach ($winemakers as $winemaker)

        <tr>

            <td>{{ $winemaker->id }}</td>

            <td>{{ $winemaker->name }}</td>

            <td>{{ $winemaker->street }} </td>

            <td>{{ $winemaker->zipcode }} {{ $winemaker->city }}</td>

            <td>{{ $winemaker->phone }}</td>

        </tr>

    @endforeach

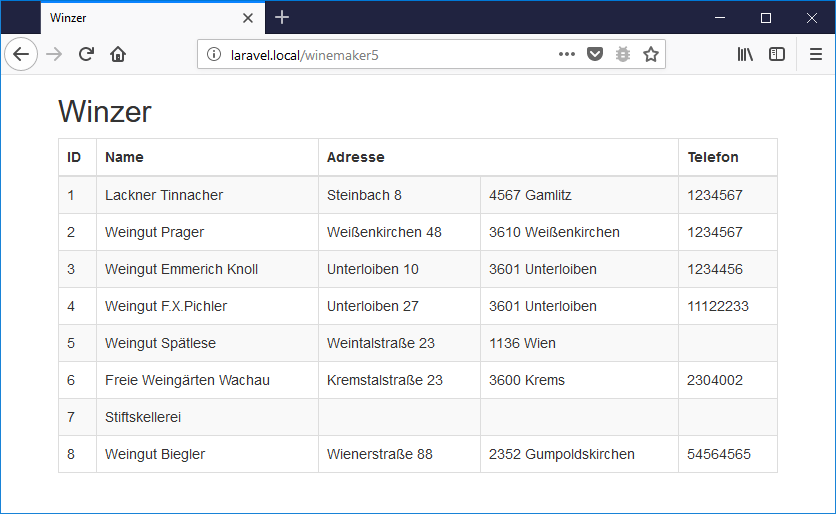
    </tbody>

</table>

</div>

</body>

</html>



CSS-Formatierung mittels tailwind (mit CDN eingebunden):

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Winzer</title>

    <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body>

<div class="container mx-auto p-4">

    <h2 class="text-2xl font-semibold mb-4">Winzer</h2>

    <table class="min-w-full bg-white border border-gray-300">

        <thead>

        <tr>

            <th class="px-4 py-2 border-b text-left">ID</th>

            <th class="px-4 py-2 border-b text-left">Name</th>

            <th class="px-4 py-2 border-b text-left" colspan="2">Adresse</th>

            <th class="px-4 py-2 border-b text-left">Telefon</th>

        </tr>

        </thead>

        <tbody>

        @foreach ($winemakers as $winemaker)

            <tr class="border-t {{ $loop->odd ? 'bg-gray-100' : '' }}">

                <td class="px-4 py-2">{{ $winemaker->id }}</td>

                <td class="px-4 py-2">{{ $winemaker->name }}</td>

                <td class="px-4 py-2">{{ $winemaker->street }}</td>

                <td class="px-4 py-2">{{ $winemaker->zipcode }} {{ $winemaker->city }}</td>

                <td class="px-4 py-2">{{ $winemaker->phone }}</td>

            </tr>

        @endforeach

        </tbody>

    </table>

</div>

</body>

</html>

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

**Routing mit Controller - mit Resource**

Route::resource('winemaker', 'WinemakerController');

erzeugt mehrere Routen gleichzeitig:

Verb Pfad Aktion Routenname

GET /winemaker index winemaker.index

GET /winemaker/create create winemaker.create

POST /winemaker store winemaker.store

GET /winemaker/{winemaker} show winemaker.show

GET /winemaker/{winemaker}/edit edit winemaker.edit

PUT/PATCH /winemaker/{winemaker} update winemaker.update

DELETE /winemaker/{winemaker} destroy winemaker.destroy

**WinemakerController:**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class WinemakerController extends Controller

{

    /\*\*

     \* Display a listing of the resource.

     \*/

    public function index()

    {

        $winemakers=\App\Models\Winemaker::all();

        return view('winemakerlist', ['winemakers'=>$winemakers]);

        //

    }

    /\*\*

     \* Show the form for creating a new resource.

     \*/

    public function create()

    {

        //

    }

    /\*\*

     \* Store a newly created resource in storage.

     \*/

    public function store(Request $request)

    {

        $validatedData = $request->validate([

            'name' => 'required',

            'street' => 'nullable',

            'zipcode' => 'nullable',

            'city' => 'nullable',

            'phone' => 'nullable'

        ]);

        $winemaker = new \App\Models\Winemaker();

        $winemaker->name = $validatedData['name'];

        $winemaker->street = $validatedData['street'];

        $winemaker->zipcode = $validatedData['zipcode'];

        $winemaker->city = $validatedData['city'];

        $winemaker->phone = $validatedData['phone'];

        $winemaker->save();

        return redirect('/winemaker')->with('status', "Winzer {$winemaker->name} hinzugefügt.");

    }

    /\*\*

     \* Display the specified resource.

     \*/

    public function show(string $id)

    {

        //

    }

    /\*\*

     \* Show the form for editing the specified resource.

     \*/

    public function edit(string $id)

    {

        $winemaker = \App\Models\Winemaker::find($id);

        return view('winemakeredit')->with('winemaker', $winemaker);

    }

    /\*\*

     \* Update the specified resource in storage.

     \*/

    public function update(Request $request, string $id)

    {

        $winemaker = \App\Models\Winemaker::find($id);

        $validatedData = $request->validate([

            'name' => 'required',

            'street' => 'nullable',

            'zipcode' => 'nullable',

            'city' => 'nullable',

            'phone' => 'nullable'

        ]);

        $winemaker->name = $validatedData['name'];

        $winemaker->street = $validatedData['street'];

        $winemaker->zipcode = $validatedData['zipcode'];

        $winemaker->city = $validatedData['city'];

        $winemaker->phone = $validatedData['phone'];

        $winemaker->save();

        //echo "Winzer {$winemaker->name} geändert.";

        return redirect('/winemaker')->with('status', "Winzer {$winemaker->name} geändert.");

    }

    /\*\*

     \* Remove the specified resource from storage.

     \*/

    public function destroy(string $id)

    {

        //

    }

    public function index2()

    {

        $winemakers=\App\Models\Winemaker::all();

        echo '<pre>';

        print\_r($winemakers);

        echo '</pre>';

    }

    public function index3()

    {

        $winemakers=\App\Models\Winemaker::all();

        return view('winemakerlistsimple3', ['winemakers'=>$winemakers]);

    }

    public function index4()

    {

        $winemakers=\App\Models\Winemaker::all();

        return view('winemakerlistsimple4', ['winemakers'=>$winemakers]);

    }

}

**winemakerlist.blade.php**

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Winzer</title>

    <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-100">

<div class="container mx-auto p-4">

    <h2 class="text-2xl font-bold mb-4">Winzer</h2>

    <table class="table-auto w-full bg-white border border-gray-300">

        <thead>

        <tr class="bg-gray-200">

            <th class="px-4 py-2 text-left">ID</th>

            <th class="px-4 py-2 text-left">Name</th>

            <th class="px-4 py-2 text-left" colspan="2">Adresse</th>

            <th class="px-4 py-2 text-left">Telefon</th>

            <th class="px-4 py-2 text-left">Action</th>

        </tr>

        </thead>

        <tbody>

        @foreach ($winemakers as $index => $winemaker)

            <tr class="{{ $index % 2 == 0 ? 'bg-gray-100' : 'bg-white' }}">

                <td class="px-4 py-2">{{ $winemaker->id }}</td>

                <td class="px-4 py-2">{{ $winemaker->name }}</td>

                <td class="px-4 py-2">{{ $winemaker->street }}</td>

                <td class="px-4 py-2">{{ $winemaker->zipcode }} {{ $winemaker->city }}</td>

                <td class="px-4 py-2">{{ $winemaker->phone }}</td>

                <td class="px-4 py-2">

                    <a href="{{ url("/winemakers/{$winemaker->id}/edit") }}" class="bg-blue-500 hover:bg-blue-700 text-white py-1 px-2 rounded-sm">Bearbeiten</a>

                </td>

            </tr>

        @endforeach

        </tbody>

    </table>

    <div class="mt-4">

        <a href="{{ url('/winemakers/create') }}" class="bg-blue-500 hover:bg-blue-700 text-white py-2 px-4 rounded">Neuer Winzer</a>

    </div>

    @if (session('status'))

        <div class="mt-4 bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded relative" role="alert">

            <span class="block sm:inline">{{ session('status') }}</span>

        </div>

    @endif

</div>

</body>

</html>

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

**winemakeredit.blade.php**

<!doctype html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Winzer bearbeiten</title>

    <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-100">

<div class="container mx-auto p-4">

    <h2 class="text-2xl mb-4">Winzer bearbeiten</h2>

    <form method="post" action="{{ url("/winemaker/{$winemaker->id}") }}">

        @csrf

        @method("put")

        <div class="mb-4">

            <label for="name" class="block mb-1">Name</label>

            <input id="name" name="name" type="text" placeholder="Name ..." class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"

                   value="{{ $winemaker->name }}">

        </div>

        <div class="mb-4">

            <label for="street" class="block mb-1">Straße</label>

            <input id="street" name="street" type="text" placeholder="Straße Hausnummer ..." class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"

                   value="{{ $winemaker->street }}">

        </div>

        <div class="mb-4">

            <label for="zipcode" class="block mb-1">Postleitzahl</label>

            <input id="zipcode" name="zipcode" type="text" placeholder="Postleitzahl ..." class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"

                   value="{{ $winemaker->zipcode }}">

        </div>

        <div class="mb-4">

            <label for="city" class="block mb-1">Ort</label>

            <input id="city" name="city" type="text" placeholder="Ort ..." class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"

                   value="{{ $winemaker->city }}">

        </div>

        <div class="mb-4">

            <label for="phone" class="block mb-1">Telefon</label>

            <input id="phone" name="phone" type="text" placeholder="Telefonnummer ..." class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"

                   value="{{ $winemaker->phone }}">

        </div>

        <div class="mb-4">

            <button type="submit" class="bg-blue-500 hover:bg-blue-700 text-white py-2 px-4 rounded">Speichern</button>

        </div>

    </form>

    <form method="post" action="{{ url("/winemaker/{$winemaker->id}") }}">

        @csrf

        @method("delete")

        <button type="submit" class="bg-red-500 hover:bg-red-700 text-white py-2 px-4 rounded">Löschen</button>

    </form>

    @if ($errors->any())

        <div class="alert alert-danger col-md-8">

            <ul>

                @foreach ($errors->all() as $error)

                    <li>{{ $error }}</li>

                @endforeach

            </ul>

        </div>

    @endif

</div>

</body>

</html>

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung