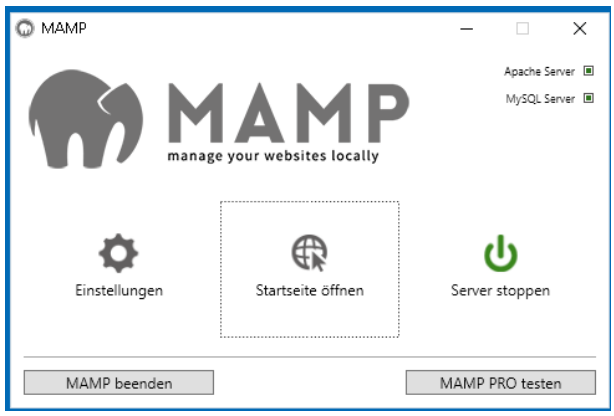
Identifizierung

Laravel bietet mit

```
php artisan make:auth
```

eine komplette BenutzerIdentifizierung mit zwei neuen Schaltflächen 'LOGIN' und 'REGISTER' in der Meüzeile

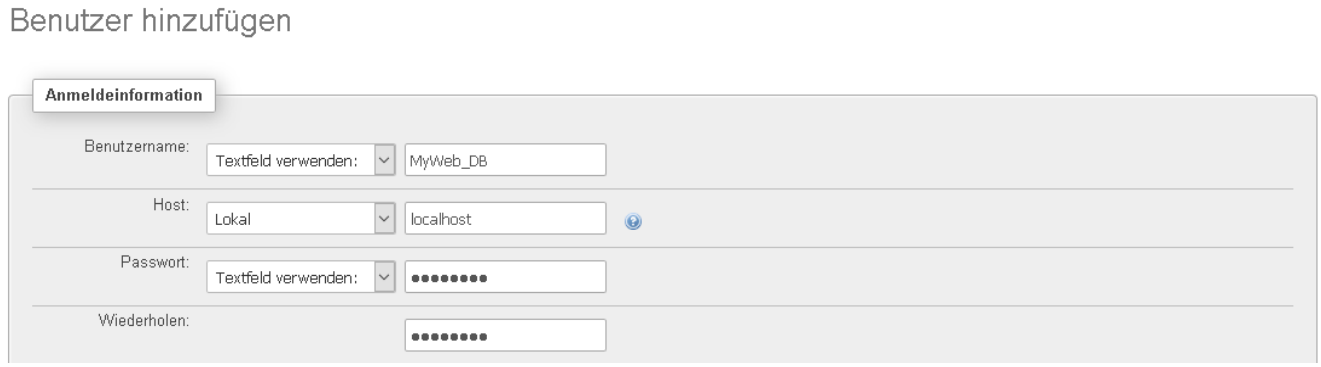


Die Identifizierung braucht eine Datenbank. Laravel unterstützt folgenden Datenbank-systeme **MySQL**[1], **PostgreSQL**[2], **SQLite**[3] und **Microsoft SQL Server**[4].

MySQL steht auf den meisten Web-Servern zur Verfügung und wird daher auch in MyWeb benutzt. Wie schon bei den Vorbereitungen angegeben, empfiehlt es sich mit MAMP zu arbeiten. Ein Klick auf 'Startseite öffnen' öffnet die Anwendung MAMP.

Mit 'phpMyAdmin' wird mit folgenden Schritten eine Datenbank MyWeb erzeugt.

Unter Benutzer kann ein neuer Benutzer (inkl. entsprechender Datenbank) hinzugefügt werden:



Dabei ist zu beachten, dass das Feld für das Erstellen einer Datenbank mit dem gleichen Namen markiert ist. Lokal wird als Passwort **MyWeb_DB** verwendet.

> ***Bitte beachten:*** *Es versteht sich von selbst, dass solche Passwörter unter keinen Umständen für den Zugriff eine Webseiter auf einem Server auf eine Datenbank verwendet werden darf.*
> *Mit Vorteil wird auf dem Server das Passwort beim Erzeugen der Datenbank von phpMyAdmin generiert.*

[1] http://www.mysql.com/

[2] https://www.postgresql.org/

[3] https://www.sqlite.org/

[4] https://www.microsoft.com/de-ch/sql-server

Es macht keinen Sinn Passwörter und Zugangsdaten offen in einer Web-Anwendung zu speichern. Diese werden in einer geschützten .env-Datei gespeichert und und von Dateien im Verzeichnis config eingelesen.

Im Fall von MyWeb muss die (lokale) .env-Datei wie folgt angepasst respektive ergänzt werden.

.env (lokal)

```
APP_NAME=MyWeb.FJKarli.ch                                    [5]
APP_ENV=local
APP_KEY=base64:njrMtqi5yg/3z+v+B/DP7dhPNPVUz50cwH+nv6WLGp0=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://localhost:81/MyWeb
APP_OWNFILES=/MyWeb/Files/                                   [6]

LOG_CHANNEL=stack


DB_CONNECTION=mysql                                          [7]
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=myweb_db                                         [8]
DB_USERNAME=MyWeb_DB
DB_PASSWORD=MyWeb_DB

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
QUEUE_DRIVER=sync

MAIL_DRIVER=smtp
MAIL_HOST=smtp.xxx                                           [9]
MAIL_PORT=2525
MAIL_USERNAME=xxx
MAIL_PASSWORD=xxx
MAIL_ENCRYPTION=null
```

festgehalten. Diese Informationen in **.env** werden mit der Funktion env() in den Dateien im Verzeichnis **config** von der Webseite übernommen.

\config\app.php

```
<?php
use Carbon\Carbon;                                           [10]
Carbon::setLocale('de');
return [
    'locale' => 'de',                                        [11]
    'ownfiles' => env('APP_OWNFILES', ''),                   [12]
];
```

Ist ein Wert in der Datei .env nicht definiert, wird das zweite Argument übernommen.

---

[5] APP_NAME wird benutzt, um die Webseite zu benennen.

[6] APP_OWNFILES wurde eingeführt, um den Speicherort für verwendete Dokumente (Bilder, PDF-Dateien, usw.) anzugeben.

[7] Wie eingangs erwähnt unterstützt Laravel MySQL, PostgreSQL, SQLite und Microsoft SQL Server. MySQL steht auf vielen Webservern zur Verfügung und wird auch hier benutzt.

[8] Die Verbindungsdaten für die lokale Datenbank MyWeb_DB wurden bewusst einfach gewählt. Es versteht sich von selbst, dass dies auf einem Webserver nicht empfohlen wird.

[9] In der Identifizierung kann sich ein registrierter Benutzer einen Password-Reset-Link zusenden lassen, falls er sein Passwort vergessen hat. Dazu braucht es Zugang zu einem Mailserver.

[10] Datum und Uhrzeit werden im englischen Format dargestellt. Carbon erlaubt dies dem eigenen Sprachraum anzupassen.

[11] Die von Laravel verwendeten englischen Systemmeldungen sind im Verzeichnis \resources\lang\en gespeichert. Falls diese in das Verzeichnis \resources\lang\de kopiert und sinngemäss übersetzt werden, können mit 'locale' => 'de' die übersetzten Meldungen verwendet werden.

[12] mit 'ownfiles' => env('APP_OWNFILES', '') wird der Speicherort von der .env-Datei übernommen,

Wie mit DB_CONNECTION angegeben wurde, kommt MySQL zum Einsatz und verwendet folgende Zugangsdaten.

\config\database.php
```
'default' => env('DB_CONNECTION', 'mysql'),
'connections' => [
...
  'mysql' => [
    'driver'    => 'mysql',
    'host'      => env('DB_HOST', 'localhost'),
    'database'  => env('DB_DATABASE', 'forge'),
    'username'  => env('DB_USERNAME', 'forge'),
    'password'  => env('DB_PASSWORD', ''),
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '',
    'strict'    => false,
  ],
...
],
```

Für die Identifizierung braucht es zwei Tabellen **users** und **password_resets**. Tabellen können von Laravel mit Migrationsdateien erstellt oder bearbeitet werden.

Migrationsdateien werden im Verzeichnis \database\migrations gespeichert. Ihr Name beginnt mit einem 'Zeitstempel' – zum Beispiel **2014_10_12_100000** - und haben einen Namen mit der Aufgabe der Migration.

Migrations-Klasse enthalten die Funktionen up und down. Die up-Funktion enthält Angaben zum Aufbau oder Änderung der Tabelle und die down-Funktion gibt an, was geschehen soll, wenn die Migration rückgängig gemacht wird. Migrationen werden mit 'migrate' ausgeführt.

Leider führt

```
php artisan migrate
```

nicht zum Erfolg. Gemäss Dokumentation [13] müssen

\database\migrations*)_create_users_table.php
```
<?php
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateUsersTable extends Migration {
  public function up() {
    Schema::create('users', function (Blueprint $table) {
      $table->increments('id');
      $table->string('name');
      $table->string('email')->users_email_unique();
      $table->string('password');
      $table->rememberToken();
      $table->timestamps();
    });
  }
  public function down() {
    Schema::dropIfExists('users');
  }
}
```
\app\Providers\AppServiceProvider.php
```
<?php namespace MyWeb\Providers;
use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Schema;
class AppServiceProvider extends ServiceProvider {
  public function boot() {
    Schema::defaultStringLength(191);
  }
  public function register() { }
}
```

angepasst werden.

[13] https://laravel-news.com/laravel-5-4-key-too-long-error

```
PS D:\Web\myweb> php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
PS D:\Web\myweb>
```

Jede Migration wird in der Tabelle 'migration' eingetragen und dabei die Nummer der Migration in der Kolonne batch eingetragen.

migration	batch
2014_10_12_000000_create_users_table	1
2014_10_12_100000_create_password_resets_table	1

Bei Bedarf könnte die jeweils letzte Migration mit **`php artisan migrate:rollback`** rückgängig gemacht werden.

Routes steuern die Verarbeitung der Links, die von Web-Anwendungen benutzt werden. Der Ablauf wird mit 'routes' gesteuert:

`\routes\web.php`

```
Route::get('/', function () { return view('welcome'); });
Auth::routes();
Route::get('/home', 'HomeController@index')->name('home');
```

Die Eingabe Http:://MyWeb/ bildet die Datei 'welcome.blade.php' ab.
Auth::routes() steuert alle Aufrufe der BenutzerIdentifizierung.
Http:://MyWeb/home ruft die Funktion index des HomeController auf. Mit

`php artisan route:list`

werden die aktiven Routes aufgelistet.

```
+--------+----------------------+-----------------+-----------------------------------------------------+--------------+
| Method | URI                  | Name            | Action                                              | Middleware   |
+--------+----------------------+-----------------+-----------------------------------------------------+--------------+
| GET|HEAD | /                    |                 | Closure                                             | web          |
| GET|HEAD | api/user             |                 | Closure                                             | api,auth:api |
| GET|HEAD | home                 | home            | closure                                             | web,auth     |
| GET|HEAD | login                | login           | Auth\LoginController@showLoginForm                  | web,guest    |
| POST   | login                |                 | Auth\LoginController@login                          | web,guest    |
| POST   | logout               | logout          | Auth\LoginController@logout                         | web          |
| POST   | password/email       | password.email  | Auth\ForgotPasswordController@sendResetLinkEmail    | web,guest    |
| GET|HEAD | password/reset       | password.request| Auth\ForgotPasswordController@showLinkRequestForm   | web,guest    |
| POST   | password/reset       |                 | Auth\ResetPasswordController@reset                  | web,guest    |
| GET|HEAD | password/reset/{token} | password.reset| Auth\ResetPasswordController@showResetForm          | web,guest    |
| GET|HEAD | register             | register        | Auth\RegisterController@showRegistrationForm        | web,guest    |
| POST   | register             |                 | Auth\RegisterController@register                    | web,guest    |
+--------+----------------------+-----------------+-----------------------------------------------------+--------------+
```

Die Steuerung einer Anwendung erfolgt über Controller im Verzeichnis \app\Http\Controllers.

`\app\Http\Controllers\HomeController.php`

```
<?php namespace App\Http\Controllers;
use Illuminate\Http\Request;
class HomeController extends Controller {
  public function __construct() { $this->middleware('auth'); }
  public function index() { return view('home'); }
}
```

Die Funktion '__contruct' aktiviert die Middleware 'auth'. Middleware erlaubt die Steuerung der Funktionen eines Controllers. Im unserem Fall wird beim Aufruf des HomeController überprüft, ob der sich der Benutzer angemeldet hat. Nur in diesem Fall werden die Funktionen des Controllers ausgeführt.

'auth' wird von der Datei Kernel.php mit Middleware gesteuert.

\app\Html\Kernel.php

```
<?php namespace App\Http;
use Illuminate\Foundation\Http\Kernel as HttpKernel;
class Kernel extends HttpKernel {
  protected $middleware = [];
  protected $middlewareGroups = ['web' => [],'api' => []];
  protected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
  ];
}
```

Views sind im Verzeichnis '\resources\views' gespeichert. Die Webseite 'MyWeb' soll nun auf dieser Basis der installierten Dateien aufgebaut werden. Um bei Bedarf auf die Originale zurückgreifen zu können, werden diese Dateien vorsichtshalber nach \resources\views\laravel kopiert.

Der Aufruf in den Routes wird nun neu:

\routes\web.php

```
Route::get('/', function () { return view('laravel.welcome'); });
Auth::routes();
Route::get('/home', 'HomeController@index')->name('home');
```

\app\Http\Controllers\HomeController.php

```
<?php namespace App\Http\Controllers;
use Illuminate\Http\Request;
class HomeController extends Controller {
  public function __construct() { $this->middleware('auth'); }
  public function index() { return view('laravel.home'); }
}
```