**Névjegyek feladat Laravel keretrendszer használatával**

**Miért Laravel?**  
Számos eszköz és keretrendszer áll rendelkezésre egy webalkalmazás létrehozásához, azonban úgy gondoljuk, hogy a Laravel a legjobb választás modern, full-stack webalkalmazások építéséhez.

**Egy Progresszív Keretrendszer**  
A Laravel-t egy „progresszív” keretrendszernek hívjuk. Ez azt jelenti, hogy a Laravel együtt fejlődik veled. Ha éppen csak most teszed meg az első lépéseket a webfejlesztés világában, a Laravel átfogó dokumentációja, útmutatói és videós anyagai segítenek megtanulni az alapokat, anélkül hogy túlterhelnének.

Ha tapasztalt fejlesztő vagy, a Laravel számos robusztus eszközt kínál a függőségkezeléshez, az egységteszteléshez, a sorok kezeléséhez, valós idejű eseményekhez, és még sok máshoz. A Laravel finomhangolt, professzionális webalkalmazások építésére alkalmas, és készen áll nagyvállalati terhelések kezelésére is.

**Egy Skálázható Keretrendszer**  
A Laravel hihetetlenül skálázható. A PHP skálázható természetének, valamint a Laravel beépített, gyors és elosztott gyorsítótár-rendszerek, mint a Redis, támogatásának köszönhetően a horizontális skálázás rendkívül egyszerű. Valójában Laravel alkalmazások könnyedén skálázhatók, hogy havi több százmillió kérést kezeljenek.

Extrém skálázásra van szükség? A Laravel Vapor platform lehetőséget nyújt, hogy a Laravel alkalmazásodat szinte korlátlan méretben futtasd az AWS legújabb, szerver nélküli technológiáján.

**Egy Közösségi Keretrendszer**  
A Laravel egyesíti a PHP ökoszisztéma legjobb csomagjait, hogy a lehető leghatékonyabb és fejlesztőbarátabb keretrendszert kínálja. Ezen felül a világ minden tájáról érkező tehetséges fejlesztők ezrei járultak hozzá a keretrendszer fejlesztéséhez. Ki tudja, talán egyszer te is a Laravel közösség hozzájáruló tagjává válsz.

**Egy Laravel Alkalmazás Létrehozása**  
**PHP és a Laravel Telepítő telepítése**  
Mielőtt elkészítenéd első Laravel alkalmazásodat, ellenőrizd, hogy a helyi gépeden telepítve van-e a PHP, a Composer és a Laravel telepítő. Emellett telepítened kell vagy a Node és NPM csomagot, vagy a Bun-t, hogy az alkalmazásod frontend elemeit is összeállíthasd.

Ha nincs telepítve a PHP és a Composer a gépeden, a következő parancsok segítségével telepítheted ezeket, valamint a Laravel telepítőt macOS, Windows vagy Linux rendszerekre.

composer global require laravel/installer

**Alkalmazás Létrehozása**  
Miután telepítetted a PHP-t, a Composer-t és a Laravel telepítőt, készen állsz egy új Laravel alkalmazás létrehozására. A Laravel telepítő kérni fogja, hogy válaszd ki a preferált tesztelési keretrendszert, az adatbázist és az indítókészletet:

laravel new example-app

Starter kit: breeze, PHPunit, dark mód yes, mysql.

Miután az alkalmazás létrejött, elindíthatod a Laravel helyi fejlesztői szervert, a sorok kezelését végző munkamenetet és a Vite fejlesztői szervert a dev Composer szkript segítségével:

npm install && npm run build

composer run dev

Amint elindítottad a fejlesztői szervert, az alkalmazás elérhető lesz a böngésződben a <http://localhost:8000> címen. Most már készen állsz arra, hogy mélyebben megismerd a Laravel ökoszisztémát. Természetesen, érdemes lehet az adatbázis beállításával is kezdened.

**Kezdeti Konfiguráció**  
A Laravel keretrendszer összes konfigurációs fájlja a config könyvtárban található. Minden opció dokumentálva van, így bátran átnézheted a fájlokat, hogy megismerd az elérhető beállításokat.

A Laravel alapvetően szinte semmilyen további konfigurációt nem igényel, így akár azonnal kezdheted a fejlesztést! Ugyanakkor érdemes átnézni a config/app.php fájlt és annak dokumentációját. Olyan beállításokat tartalmaz, mint az időzóna és a nyelvi beállítás, amelyeket az alkalmazásodhoz igazíthatsz.

A config/app.php fájl a Laravel alkalmazás általános beállításait tartalmazza, melyeket érdemes áttekinteni és az igényekhez igazítani. Lássuk részletesen az egyes kulcsokat és azok funkcióit:

1. **Name**  
   Az alkalmazás neve, amit az env('APP\_NAME', 'Laravel') sor állít be. Az .env fájl APP\_NAME változójával szabható testre.

'name' => env('APP\_NAME', 'Laravel'),

1. **Environment**  
   Az alkalmazás környezetét határozza meg (pl. local, production). Ez a környezeti változók alapján kerül beállításra az .env fájl APP\_ENV változójában.

'env' => env('APP\_ENV', 'production'),

1. **Debug Mode**  
   A hibakeresési mód (true vagy false), ami meghatározza, hogy a részletes hibajelentések megjelenjenek-e. Az .env fájl APP\_DEBUG változójával kezelhető.

'debug' => env('APP\_DEBUG', false),

1. **URL**  
   Az alkalmazás alapértelmezett URL-je, amit az APP\_URL változóval az .env fájlban adhatunk meg. Ezt a rendszer számos helyen használja, például hivatkozások és külső hívások során.

'url' => env('APP\_URL', 'http://localhost'),

1. **Timezone**  
   Az időzóna, amit az alkalmazás használ. Ezt Europe/Budapest-re lehet állítani Magyarország esetén, vagy más tetszőleges időzónára.

'timezone' => 'UTC',

1. **Locale**  
   Az alapértelmezett nyelvi beállítás (pl. en az angol, hu a magyar). Ez az alkalmazás lokalizációjához használatos.

'locale' => 'en',

1. **Fallback Locale**  
   A tartalék nyelv, amelyet akkor használ az alkalmazás, ha a kiválasztott nyelv nem elérhető.

'fallback\_locale' => 'en',

1. **Key**  
   Az alkalmazás titkosítási kulcsa. Ezt az APP\_KEY változó tartalmazza az .env fájlban, és elengedhetetlen a biztonságos titkosításhoz. A kulcs generálásához a php artisan key:generate parancsot használhatjuk.

'key' => env('APP\_KEY'),

1. **Cipher**  
   A titkosítás algoritmusa. Laravel az AES-256-CBC vagy AES-128-CBC titkosítást támogatja.

'cipher' => 'AES-256-CBC',

Az **AES-256-CBC** és **AES-128-CBC** titkosítások az **Advanced Encryption Standard (AES)** módszert használó, blokkalapú titkosítási algoritmusok. Az AES egy szabványos titkosítási algoritmus, amelyet az amerikai Nemzeti Szabványügyi és Technológiai Intézet (NIST) fejlesztett ki, és gyakran használják különféle adatbiztonsági megoldásokban.

A két fő jellemzőjük:

1. **AES-256 és AES-128**  
   Az algoritmus neve utáni szám (256 vagy 128) az úgynevezett "kulcshossz" hossza, amely az algoritmus által használt kulcs méretét jelzi **bitekben**. A kulcshossz befolyásolja a titkosítás erősségét és biztonságát:
   * **AES-128**: 128 bites kulcs, gyorsabb, kevesebb memóriát használ, és biztonságosnak tekinthető a legtöbb célra.
   * **AES-256**: 256 bites kulcs, nagyobb biztonságot nyújt, bár kicsit lassabb és több memóriát igényel.

A 256 bites kulcshosszú titkosítás sokkal nehezebb feltörni, mivel több lehetséges kulcsot tartalmaz, így nagyobb számítási erőforrást igényel a visszafejtése.

1. **CBC (Cipher Block Chaining)**  
   A **CBC** a titkosítási mód, amely azt határozza meg, hogyan titkosítja az AES algoritmus a blokkokat:
   * A CBC módban a titkosítás az előző blokk titkosított eredményére (ciphertext) alapozza a következő blokk titkosítását, így egy láncolás (chaining) jön létre.
   * Az első blokkhoz egy ún. **kezdő érték** (IV, Initialization Vector) szükséges, hogy minden titkosítás egyedi legyen. Az IV biztosítja, hogy még ugyanazt az adatot is más-más kódolt eredménnyel tárolja.

A CBC móddal kombinált AES-256 vagy AES-128 kiváló választás adatbiztonsági célokra, mivel erős titkosítást biztosít, és a kezdő értékek használatával gondoskodik arról, hogy a titkosított adatok ne ismétlődjenek.

1. **Providers**  
   Az alkalmazás által használt szolgáltatókat (service providers) sorolja fel. Ezek betöltik a Laravel keretrendszer összes fontos szolgáltatását, mint például az adatbázis-kezelést és az autentikációt. Egyéni szolgáltatók hozzáadhatók a listához szükség esetén.

'providers' => [

/\*

\* Laravel Framework Service Providers...

\*/

Illuminate\Auth\AuthServiceProvider::class,

Illuminate\Broadcasting\BroadcastServiceProvider::class,

// ...

],

1. **Aliases**  
   Rövidítések (aliasok) vagy "álnév" lista, amely gyors hozzáférést biztosít a gyakran használt osztályokhoz, mint például Route, DB, Schema. Így ezeket a rövid neveket egyszerűen lehet használni a kódban a teljes névtér helyett.

'aliases' => [

'App' => Illuminate\Support\Facades\App::class,

'DB' => Illuminate\Support\Facades\DB::class,

// ...

],

1. **Log Settings**  
   Laravel naplózási beállításai. Az .env fájlban az LOG\_CHANNEL változó szabályozza, hogy a rendszer milyen naplózási csatornát használjon (például: single, stack, daily, syslog, errorlog).

'log' => env('LOG\_CHANNEL', 'stack'),

1. **Mail Settings**  
   A Laravel alkalmazás e-mail küldési beállításait az config/mail.php fájlban részletezheted, és az .env fájlban megadhatók az SMTP-beállítások (például MAIL\_MAILER, MAIL\_HOST, MAIL\_PORT, stb.).

Ez a fájl tehát központi szerepet játszik a Laravel alkalmazás általános működésében. Ha ezeknek a beállításoknak a pontos célját és működését jól átlátod, könnyebben fogod tudni az alkalmazásodat az egyéni igényekhez igazítani.

**Környezeti Alapú Konfiguráció**  
A Laravel számos konfigurációs opciójának értéke változhat attól függően, hogy az alkalmazásod helyi gépen vagy éles környezetben fut-e. Ezért sok fontos konfigurációs érték a projekt gyökérkönyvtárában található .env fájlban van megadva.

A .env fájlt nem szabad a forráskódkezelőbe feltölteni, mivel minden fejlesztőnek vagy szervernek saját környezeti konfigurációra lehet szüksége. Emellett biztonsági kockázatot jelentene, ha egy illetéktelen hozzáférne a forráskód-tárolóhoz, és így érzékeny adatokhoz jutna.

További információért a .env fájlról és a környezeti alapú konfigurációról olvasd el a teljes konfigurációs dokumentációt.

**Adatbázisok és Migrációk**  
Most, hogy létrehoztad a Laravel alkalmazást, valószínűleg adatokat szeretnél tárolni egy adatbázisban. Az alkalmazás .env konfigurációs fájljában alapértelmezés szerint egy SQLite adatbázis van megadva, amelyhez a Laravel csatlakozni fog.

Az alkalmazás létrehozásakor a Laravel automatikusan létrehozott egy database/database.sqlite fájlt, és lefuttatta a szükséges migrációkat az adatbázis tábláinak létrehozásához.

Ha más adatbázis-kezelőt szeretnél használni, például MySQL-t vagy PostgreSQL-t, akkor frissítheted az .env konfigurációs fájlt a megfelelő adatbázis beállításokkal. Például, ha MySQL-t szeretnél használni, frissítsd az .env fájl DB\_\* változóit az alábbi módon:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=nevjegyek

DB\_USERNAME=root

DB\_PASSWORD=

SESSION\_DRIVER=file //de lehet a database-t is használni

Ha a session-nél az adatbázist akarod használni:

php artisan session:table

php artisan migrate

Most, hogy a nevjegyek táblát csatoltad a Laravel projekthez, a következő lépésekben használhatod azt:

alkalmazkodjunk az adatbázis meglévő szerkezetéhez és adataihoz, és hozzunk létre egy modellt és vezérlőt, amelyek a meglévő nevjegyek táblát használják migrációk nélkül.

### 1. Nevjegy ****Model Létrehozása****

Hozz létre egy modellt a nevjegyek tábla kezelésére:

php artisan make:model Nevjegy

Ez létrehoz egy Nevjegy.php modellt az app/Models mappában.

Szerkeszd a modellt, hogy megfelelően illeszkedjen a táblához:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Nevjegy extends Model

{

// A tábla neve

protected $table = 'nevjegyek';

// A mezők, amelyeket tömegesen ki lehet tölteni

protected $fillable = ['foto', 'nev', 'cegnev', 'foglalkozas', 'email', 'mobil'];

}

### 2. ****Vezérlő Létrehozása a Névjegyek Kezelésére****

Hozz létre egy vezérlőt a CRUD (Create, Read, Update, Delete) műveletekhez:

php artisan make:controller NevjegyController

Ez létrehoz egy NevjegyController.php fájlt az app/Http/Controllers mappában.

A vezérlőben definiálhatunk metódusokat az adatbázisban lévő nevjegyek tábla adatainak megjelenítésére, létrehozására, frissítésére és törlésére.

### 3. index ****Metódus az Összes Névjegy Listázására****

Az index metódus az összes névjegyet lekérdezi a nevjegyek táblából és átadja a nézetnek.

A NevjegyController-ben definiáld az index metódust:

use App\Models\Nevjegy;

public function index()

{

$nevjegyek = Nevjegy::all(); // Az összes névjegy lekérése

return view('nevjegyek.index', compact('nevjegyek'));

}

### 4. ****Útvonal Hozzáadása a**** web.php ****Fájlban****

A routes/web.php fájlban hozz létre egy útvonalat a NevjegyController index metódusához:

use App\Http\Controllers\NevjegyController;

Route::get('/nevjegyek', [NevjegyController::class, 'index'])->name('nevjegyek.index');

### 5. ****Nézet Létrehozása az Adatok Megjelenítéséhez****

Hozz létre egy nézetfájlt a resources/views/nevjegyek/index.blade.php útvonalon, amely megjeleníti a névjegyeket.

@extends('layouts.app')

@section('content')

<h1>Névjegyek</h1>

<table>

<tr>

<th>Név</th>

<th>Cégnév</th>

<th>Foglalkozás</th>

<th>Email</th>

<th>Mobil</th>

</tr>

@foreach($nevjegyek as $nevjegy)

<tr>

<td>{{ $nevjegy->nev }}</td>

<td>{{ $nevjegy->cegnev }}</td>

<td>{{ $nevjegy->foglalkozas }}</td>

<td>{{ $nevjegy->email }}</td>

<td>{{ $nevjegy->mobil }}</td>

</tr>

@endforeach

</table>

@endsection

### Hozz létre egy layouts nevű mappát a nézetek könyvtárában, és benne egy app.blade.php fájlt:

Nyisd meg a app.blade.php fájlt, és add hozzá az alábbi kódot, amely az alap HTML struktúrát és egy @yield('content') helyet biztosít a tartalom számára:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@yield('title', 'Névjegyek App')</title>

<link rel="stylesheet" href="{{ asset('css/app.css') }}">

</head>

<body>

<header>

<h1>Laravel Névjegyek App</h1>

<nav>

<a href="{{ route('nevjegyek.index') }}">Névjegyek</a>

<!-- Egyéb navigációs linkek -->

</nav>

</header>

<main>

@yield('content')

</main>

<footer>

<p>&copy; {{ date('Y') }} Névjegyek App</p>

</footer>

</body>

</html>

### 6. ****Gyorsítótár Törlése és Szerver Újraindítása****

Ha készen állsz, futtasd az alábbi parancsokat a gyorsítótár tisztításához és a szerver újraindításához:

php artisan config:clear

php artisan route:clear

php artisan view:clear

php artisan serve

Íme a böngészőben a <http://127.0.0.1:8000/nevjegyek> url címen megjelenik az összes névjegy egy táblázatban, de mi nem így szeretnénk, hiszen ehhez volt már egy nézetünk.

először is a app\Http\Controllers\NevjegyController.php fájlt ki kell egészíteni egy új metódussal:

public function welcome(Request $request)

{

$kifejezes = $request->input('kifejezes', '');

$mennyit = 9;

$nevjegyek = Nevjegy::where('nev', 'like', "%{$kifejezes}%")

->orWhere('cegnev', 'like', "%{$kifejezes}%")

->orWhere('mobil', 'like', "%{$kifejezes}%")

->orWhere('email', 'like', "%{$kifejezes}%")

->orderBy('nev', 'asc')

->paginate($mennyit)

->withQueryString();

return view('welcome', compact('nevjegyek', 'kifejezes'));

}

meg kell adnunk az útvonalat:

Route::get('/', [NevjegyController::class, 'welcome'])->name('welcome');

//login

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

return view('dashboard');

})->middleware(['auth'])->name('dashboard');

**use App\Http\Controllers\ProfileController;**

Route::middleware('auth')->group(function () { Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit'); Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update'); Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy'); });

require \_\_DIR\_\_.'/auth.php';

eredeti welcome-ot írd át erre.

### dashboard.blade.php:

@extends('layouts.app')

@section('content')

<h1>Üdvözlünk a Dashboardon, {{ Auth::user()->name }}!</h1>

@if (Auth::user()->is\_admin)

<div class="admin-menu">

<h2>Admin Műveletek</h2>

<ul>

<li><a href="{{ route('admin.nevjegyek.create') }}">Új névjegy hozzáadása</a></li>

<li><a href="{{ route('admin.nevjegyek.index') }}">Névjegyek kezelése</a></li>

</ul>

</div>

@endif

@endsection

Győződj meg róla, hogy a stílusok elérhetők. A public/css/app.css fájlba beállíthatod a saját stílusaidat, amelyeket a welcome.blade.php-nél használsz, vagyis, hozd létre a megadott útvonalon az app.css fájlt és másold bele a régi feladatból a css-t.

Helyezd el a kepek mappát a public könyvtárban, hogy a képek elérhetők legyenek a public/kepek útvonalon keresztül. A Laravel így a {{ asset('kepek/' . $nevjegy->foto) }} útvonalat használva éri el a képeket.

El kell készíteni a belépési lehetőséget is majd az adminnak, ehhez használjuk a breeze-t

composer require laravel/breeze –dev

php artisan breeze:install

php artisan migrate

Ezzel a welcome.blade.php megjeleníti a névjegyeket, lehetőséget biztosít keresésre, és a lapozóval is működni fog.

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Névjegykártyák</title>

<link rel="stylesheet" href="{{ asset('css/app.css') }}">

</head>

<body>

<header>

<h1>Névjegykártyák</h1>

<nav>

<a href="{{ route('login') }}">Bejelentkezés</a>

<a href="{{ route('register') }}">Regisztráció</a>

</nav>

</header>

@extends('layouts.app')

@section('content')

<h1>Névjegykártyák</h1>

<form method="GET" action="{{ route('welcome') }}">

<input type="search" id="kifejezes" name="kifejezes" placeholder="Keresés..." value="{{ $kifejezes }}">

<button type="submit">Keresés</button>

</form>

@if ($nevjegyek->isEmpty())

<article>

<h2>Nincs találat a rendszerben!</h2>

</article>

@else

<div class="container">

@foreach ($nevjegyek as $nevjegy)

<article>

<img src="{{ asset('kepek/' . $nevjegy->foto) }}" alt="{{ $nevjegy->nev }}">

<h2>{{ $nevjegy->nev }}</h2>

<h3>{{ $nevjegy->cegnev }}</h3>

<p>Mobil: <a href="tel:{{ $nevjegy->mobil }}">{{ $nevjegy->mobil }}</a></p>

<p>E-mail: <a href="mailto:{{ $nevjegy->email }}">{{ $nevjegy->email }}</a></p>

</article>

@endforeach

</div>

{{ $nevjegyek->links() }} <!-- Lapozó linkek -->

@endif

@endsection

Nos, van egy welcome oldalunk, amit mindenki láthat, és egy nevjegyek/index ami táblázatos, oldjuk meg, hogy ezt a táblázatot csak az admin láthassa.

### 1. is\_admin ****Oszlop Hozzáadása a**** users ****Táblához****

Hozzunk létre egy migrációt az is\_admin oszlop hozzáadásához, amely jelezni fogja, hogy egy felhasználó admin-e vagy sem.

php artisan make:migration add\_is\_admin\_to\_users\_table --table=users

Nyisd meg a létrejött migrációt a database/migrations könyvtárban, és add hozzá az alábbi kódot:

public function up()

{

Schema::table('users', function (Blueprint $table) {

**$table->boolean('is\_admin')->default(0)->after('password'); // 0 =** nem admin, 1 = admin

});

}

public function down()

{

Schema::table('users', function (Blueprint $table) {

**$table->dropColumn('is\_admin');**

});

}

Ezután futtasd a migrációt:

php artisan migrate

Ha most megnézed az adatbázisodban a users táblát, kiegészült egy oszloppal, az is\_adminnal ahol alapértelmezetten 0 lesz a beállítás, de ha kézzel átállítjuk 1-re akkor a regisztrált user admin is lesz.

### 2. ****Admin Middleware Létrehozása****

Hozzunk létre egy middleware-t, amely ellenőrzi, hogy a felhasználó admin-e. Ha nem admin, akkor átirányítjuk őt egy másik oldalra.

php artisan make:middleware AdminMiddleware

Szerkeszd a AdminMiddleware.php fájlt az app/Http/Middleware könyvtárban:

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class AdminMiddleware

{

public function handle(Request $request, Closure $next)

{

if (Auth::check() && Auth::user()->is\_admin) {

return $next($request);

}

return redirect('/')->with('error', 'Nincs jogosultságod az oldal megtekintéséhez.');

}

}

Ezután regisztráld az AdminMiddleware-t a nevjegyekcontroller-t ki kell egészíteni:

<?php

namespace App\Http\Controllers;

use App\Models\Nevjegy;

use Illuminate\Http\Request;

use Illuminate\Routing\Controllers\HasMiddleware;

use Illuminate\Routing\Controllers\Middleware;

class NevjegyController extends Controller implements HasMiddleware

{

public static function middleware(): array

{

return [

new Middleware(\App\Http\Middleware\AdminMiddleware::class, only: ['index', 'create', 'edit', 'update', 'destroy']),

];

}

public function welcome(Request $request)

{

$kifejezes = $request->input('kifejezes', '');

$mennyit = 9;

$nevjegyek = Nevjegy::where('nev', 'like', "%{$kifejezes}%")

->orWhere('cegnev', 'like', "%{$kifejezes}%")

->orWhere('mobil', 'like', "%{$kifejezes}%")

->orWhere('email', 'like', "%{$kifejezes}%")

->orderBy('nev', 'asc')

->paginate($mennyit)

->withQueryString();

return view('welcome', compact('nevjegyek', 'kifejezes'));

}

public function index()

{

$nevjegyek = Nevjegy::all(); // Az összes névjegy lekérése

return view('nevjegyek.index', compact('nevjegyek'));

}

}

### 3. ****Útvonalak Beállítása az Admin Műveletekhez****

A routes/web.php fájlban adj hozzá útvonalakat az admin számára a nevjegyek kezelésére:

use App\Http\Controllers\NevjegyController;

Route::get('/', [NevjegyController::class, 'welcome'])->name('welcome');

Route::middleware(['auth', 'admin'])->get('/admin/nevjegyek', [NevjegyController::class, 'index'])->name('admin.nevjegyek.index');

Route::middleware(['auth', 'admin'])->group(function () {

Route::get('/admin/nevjegyek', [NevjegyController::class, 'index'])->name('admin.nevjegyek.index');

Route::get('/admin/nevjegyek/create', [NevjegyController::class, 'create'])->name('admin.nevjegyek.create');

Route::post('/admin/nevjegyek', [NevjegyController::class, 'store'])->name('admin.nevjegyek.store');

Route::get('/admin/nevjegyek/{id}/edit', [NevjegyController::class, 'edit'])->name('admin.nevjegyek.edit');

Route::put('/admin/nevjegyek/{id}', [NevjegyController::class, 'update'])->name('admin.nevjegyek.update');

Route::delete('/admin/nevjegyek/{id}', [NevjegyController::class, 'destroy'])->name('admin.nevjegyek.destroy');

});

### 4. ****Nézetek Létrehozása:**** index****,**** create****,**** edit****, és**** delete

Hozz létre fájlokat a resources/views/nevjegyek könyvtárban: index.blade.php, create.blade.php, edit.blade.php, és egy törlés megerősítő felületet (delete.blade.php), ahol a műveletek megtörténnek.

### index.blade.php

Ez az oldal táblázatos formában listázza a névjegyeket, és csak az admin láthatja.

@extends('layouts.app')

@section('content')

<h2>Névjegyek kezelése</h2>

<a href="{{ route('admin.nevjegyek.create') }}">Új névjegy hozzáadása</a>

<table>

<tr>

<th>Név</th>

<th>Cégnév</th>

<th>Email</th>

<th>Mobil</th>

<th>Műveletek</th>

</tr>

@foreach($nevjegyek as $nevjegy)

<tr>

<td>{{ $nevjegy->nev }}</td>

<td>{{ $nevjegy->cegnev }}</td>

<td>{{ $nevjegy->email }}</td>

<td>{{ $nevjegy->mobil }}</td>

<td>

<a href="{{ route('admin.nevjegyek.edit', $nevjegy->id) }}">Szerkesztés</a>

<form action="{{ route('admin.nevjegyek.destroy', $nevjegy->id) }}" method="POST" style="display:inline;">

@csrf

@method('DELETE')

<button type="submit">Törlés</button>

</form>

</td>

</tr>

@endforeach

</table>

@endsection

### create.blade.php

Form névjegy hozzáadásához, képfeltöltési lehetőséggel.

@extends('layouts.app')

@section('content')

<h2>Új névjegy hozzáadása</h2>

<form action="{{ route('admin.nevjegyek.store') }}" method="POST" enctype="multipart/form-data">

@csrf

<div>

<label for="nev">Név:</label>

<input type="text" id="nev" name="nev" required>

</div>

<div>

<label for="cegnev">Cégnév:</label>

<input type="text" id="cegnev" name="cegnev">

</div>

<div>

<label for="foglalkozas">Foglalkozás:</label>

<input type="text" id="foglalkozas" name="foglalkozas">

</div>

<div>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

</div>

<div>

<label for="mobil">Mobil:</label>

<input type="text" id="mobil" name="mobil">

</div>

<div>

<label for="foto">Kép feltöltése:</label>

<input type="file" id="foto" name="foto">

</div>

<button type="submit">Mentés</button>

</form>

@endsection

### edit.blade.php

Form meglévő névjegy szerkesztésére.

@extends('layouts.app')

@section('content')

<div class="container">

<h1>Névjegy szerkesztése</h1>

{{-- Hibák megjelenítése --}}

@if ($errors->any())

<div class="alert alert-danger">

<ul>

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

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

@endforeach

</ul>

</div>

@endif

{{-- Névjegy szerkesztési űrlap --}}

<form action="{{ route('admin.nevjegyek.update', $nevjegy->id) }}" method="POST" enctype="multipart/form-data">

@csrf

@method('PUT')

{{-- Név mező --}}

<div class="form-group">

<label for="nev">Név</label>

<input type="text" name="nev" id="nev" class="form-control" value="{{ old('nev', $nevjegy->nev) }}" required>

</div>

{{-- Cégnév mező --}}

<div class="form-group">

<label for="cegnev">Cégnév</label>

<input type="text" name="cegnev" id="cegnev" class="form-control" value="{{ old('cegnev', $nevjegy->cegnev) }}">

</div>

{{-- Mobil mező --}}

<div class="form-group">

<label for="mobil">Mobil</label>

<input type="text" name="mobil" id="mobil" class="form-control" value="{{ old('mobil', $nevjegy->mobil) }}">

</div>

{{-- E-mail mező --}}

<div class="form-group">

<label for="email">E-mail</label>

<input type="email" name="email" id="email" class="form-control" value="{{ old('email', $nevjegy->email) }}">

</div>

{{-- Kép megjelenítése és feltöltési mező --}}

<div class="form-group">

<label>Jelenlegi Fotó</label>

<div>

<img src="{{ asset('storage/kepek/' . $nevjegy->foto) }}" alt="Fotó" width="150">

</div>

</div>

<div class="form-group">

<label for="foto">Új Fotó (ha módosítani szeretnéd)</label>

<input type="file" name="foto" id="foto" class="form-control">

</div>

{{-- Mentés gomb --}}

<button type="submit" class="btn btn-primary">Mentés</button>

</form>

</div>

@endsection

### delete.blade.php

Megerősítő oldal törléshez.

### Útvonalak hozzáadása:

### Route::middleware(['auth', AdminMiddleware::class])->prefix('admin/nevjegyek')->name('admin.nevjegyek.')->group(function () {

### Route::get('/', [NevjegyController::class, 'index'])->name('index');

### Route::get('/create', [NevjegyController::class, 'create'])->name('create');

### Route::post('/', [NevjegyController::class, 'store'])->name('store');

### Route::get('/{id}/edit', [NevjegyController::class, 'edit'])->name('edit');

### Route::put('/{id}', [NevjegyController::class, 'update'])->name('update');

### Route::delete('/{id}', [NevjegyController::class, 'destroy'])->name('destroy');

### });5. ****NevjegyController CRUD Műveletek Készítése****

A NevjegyController-ben definiálj metódusokat a fenti műveletekhez, és biztosítsd, hogy a fájlfeltöltés megfelelően történjen. Ha konkrét segítségre van szükség ezekhez a metódusokhoz, részletesen le tudjuk írni őket is.

public function create()

{

return view('nevjegyek.create');

}

public function store(Request $request)

{

// Adatmentés logika

}

public function edit($id)

{

$nevjegy = Nevjegy::findOrFail($id);

return view('nevjegyek.edit', compact('nevjegy'));

}

public function update(Request $request, $id)

{

// Adatfrissítés logika

}

public function destroy($id)

{

$nevjegy = Nevjegy::findOrFail($id);

$nevjegy->delete();

return redirect()->route('admin.nevjegyek.index')->with('success', 'Névjegy törölve.');

}

### Na már most jó lenne ha táblázatot kiegészítenénk törléssel és módosítással, és rendezéssel, de a rendezéshez módosítani kell először a nevjegyekcontrollert:

### <?php

### namespace App\Http\Controllers;

### use App\Models\Nevjegy;

### use Illuminate\Http\Request;

### use App\Http\Middleware\AdminMiddleware;

### class NevjegyController extends Controller

### {

### 

### public function welcome(Request $request)

### {

### $kifejezes = $request->input('kifejezes', '');

### $mennyit = 9;

### $nevjegyek = Nevjegy::where('nev', 'like', "%{$kifejezes}%")

### ->orWhere('cegnev', 'like', "%{$kifejezes}%")

### ->orWhere('mobil', 'like', "%{$kifejezes}%")

### ->orWhere('email', 'like', "%{$kifejezes}%")

### ->orderBy('nev', 'asc')

### ->paginate($mennyit)

### ->withQueryString();

### return view('welcome', compact('nevjegyek', 'kifejezes'));

### }

### public function index(Request $request)

### {

### $kifejezes = $request->input('kifejezes', ''); // Keresési kifejezés

### $mennyit = 9; // Lapozás mértéke

### // Névjegyek lekérése szűrés és rendezés alapján

### $nevjegyek = Nevjegy::where('nev', 'like', "%{$kifejezes}%")

### ->orWhere('cegnev', 'like', "%{$kifejezes}%")

### ->orWhere('mobil', 'like', "%{$kifejezes}%")

### ->orWhere('email', 'like', "%{$kifejezes}%")

### ->orderBy('nev', 'asc')

### ->paginate($mennyit)

### ->withQueryString();

### return view('nevjegyek.index', compact('nevjegyek', 'kifejezes'));

### }

### public function create()

### {

### return view('nevjegyek.create');

### }

### public function store(Request $request)

### {

### $request->validate([

### 'nev' => 'required|string|max:255',

### 'cegnev' => 'nullable|string|max:255',

### 'mobil' => 'nullable|string|max:20',

### 'email' => 'nullable|email|max:255',

### 'foto' => 'nullable|image|mimes:jpg,jpeg,png|max:2048',

### ]);

### $nevjegy = new Nevjegy();

### $nevjegy->nev = $request->input('nev');

### $nevjegy->cegnev = $request->input('cegnev');

### $nevjegy->mobil = $request->input('mobil');

### $nevjegy->email = $request->input('email');

### if ($request->hasFile('foto')) {

### $fotoPath = $request->file('foto')->store('kepek', 'public');

### $nevjegy->foto = basename($fotoPath);

### } else {

### $nevjegy->foto = 'nincskep.png';

### }

### $nevjegy->save();

### return redirect()->route('admin.nevjegyek.index')->with('success', 'Névjegy sikeresen hozzáadva.');

### }

### public function edit($id)

### {

### $nevjegy = Nevjegy::findOrFail($id);

### return view('nevjegyek.edit', compact('nevjegy'));

### }

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

### {

### $request->validate([

### 'nev' => 'required|string|max:255',

### 'cegnev' => 'nullable|string|max:255',

### 'mobil' => 'nullable|string|max:20',

### 'email' => 'nullable|email|max:255',

### 'foto' => 'nullable|image|mimes:jpg,jpeg,png|max:2048',

### ]);

### $nevjegy = Nevjegy::findOrFail($id);

### $nevjegy->nev = $request->input('nev');

### $nevjegy->cegnev = $request->input('cegnev');

### $nevjegy->mobil = $request->input('mobil');

### $nevjegy->email = $request->input('email');

### if ($request->hasFile('foto')) {

### if ($nevjegy->foto && $nevjegy->foto != 'nincskep.png') {

### Storage::disk('public')->delete('kepek/' . $nevjegy->foto);

### }

### $fotoPath = $request->file('foto')->store('kepek', 'public');

### $nevjegy->foto = basename($fotoPath);

### }

### $nevjegy->save();

### return redirect()->route('admin.nevjegyek.index')->with('success', 'Névjegy sikeresen frissítve.');

### }

### public function destroy($id)

### {

### $nevjegy = Nevjegy::findOrFail($id);

### if ($nevjegy->foto && $nevjegy->foto != 'nincskep.png') {

### Storage::disk('public')->delete('kepek/' . $nevjegy->foto);

### }

### $nevjegy->delete();

### return redirect()->route('admin.nevjegyek.index')->with('success', 'Névjegy törölve.');

### }

### }

### }

### és a nevjegyek/index

@extends('layouts.app')

@section('content')

<h1>Névjegykártyák</h1>

<!-- Keresőmező -->

<form method="GET" action="{{ route('admin.nevjegyek.index') }}">

<input type="search" id="kifejezes" name="kifejezes" value="{{ $kifejezes }}" placeholder="Keresés...">

<button type="submit">Keresés</button>

</form>

<!-- Új névjegy hozzáadása és kilépés -->

<p>

<a href="{{ route('admin.nevjegyek.create') }}">Új névjegy hozzáadása</a> |

<a href="{{ route('logout') }}" onclick="event.preventDefault(); document.getElementById('logout-form').submit();">Kilépés</a>

</p>

<form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">

@csrf

</form>

<!-- Táblázat -->

<table>

<thead>

<tr>

<th>Fotó</th>

<th><a href="{{ route('admin.nevjegyek.index', ['rendez' => 'nev']) }}">Név</a></th>

<th><a href="{{ route('admin.nevjegyek.index', ['rendez' => 'cegnev']) }}">Cégnév</a></th>

<th><a href="{{ route('admin.nevjegyek.index', ['rendez' => 'mobil']) }}">Mobil</a></th>

<th><a href="{{ route('admin.nevjegyek.index', ['rendez' => 'email']) }}">E-mail</a></th>

<th>Műveletek</th>

</tr>

</thead>

<tbody>

@forelse ($nevjegyek as $nevjegy)

<tr>

<td <img src="{{ asset('storage/kepek/' . ($nevjegy->foto ?? 'nincskep.png')) }}" alt="{{ $nevjegy->nev }}" width="50"></td>

<td>{{ $nevjegy->nev }}</td>

<td>{{ $nevjegy->cegnev }}</td>

<td>{{ $nevjegy->mobil }}</td>

<td>{{ $nevjegy->email }}</td>

<td>

<a href="{{ route('admin.nevjegyek.edit', $nevjegy->id) }}">Módosítás</a> |

<form action="{{ route('admin.nevjegyek.destroy', $nevjegy->id) }}" method="POST" style="display:inline;">

@csrf

@method('DELETE')

<button type="submit" onclick="return confirm('Biztosan törölni szeretnéd?')">Törlés</button>

</form>

</td>

</tr>

@empty

<tr>

<td colspan="6">Nincs találat a rendszerben!</td>

</tr>

@endforelse

</tbody>

</table>

<!-- Lapozó -->

{{ $nevjegyek->links() }}

@endsection

### Timestamps Kezelésének Kikapcsolása a Nevjegy Modellben

Ha megpróbálunk adatot felvinni hibaüzenetet kapunk, mert időbélyeg is került az adatbázisba, ha ezeket ki szeretnénk tiltani módosítani kell a Nevjegy modellt, és adj hozzá egy public $timestamps = false; sort, hogy letiltsd az automatikus időbélyegzőket:

class Nevjegy extends Model

{

public $timestamps = false; // Időbélyegek kikapcsolása

// A tábla neve

protected $table = 'nevjegyek';

// A mezők, amelyeket tömegesen ki lehet tölteni

protected $fillable = ['foto', 'nev', 'cegnev', 'foglalkozas', 'email', 'mobil'];

}

### Nézzük meg mi történik, ha az adatbázisodat rosszul tervezted meg, és mondjuk a telefonszám nem fér el a mezőben:

Megoldás: Az Adatbázis Módosítása

A mobil oszlop méretének növeléséhez frissítenünk kell az adatbázis sémát.

1. Módosító Migráció Készítése

Futtasd az alábbi parancsot egy új migráció létrehozásához, amely frissíti a mobil oszlop hosszát:

php artisan make:migration alter\_mobil\_column\_in\_nevjegyek\_table --table=nevjegyek

2. Migráció Szerkesztése

Nyisd meg az újonnan létrehozott migrációs fájlt a database/migrations mappában, és módosítsd a mobil oszlop hosszát, például 20 karakterre (vagy amennyi elegendő lesz):

**database\migrations\2024\_10\_30\_193404\_alter\_mobil\_column\_in\_nevjegyek\_table.php**

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class AlterMobilColumnInNevjegyekTable extends Migration

{

public function up()

{

Schema::table('nevjegyek', function (Blueprint $table) {

$table->string('mobil', 20)->change(); // Növeljük a hosszát 20-ra

});

}

public function down()

{

Schema::table('nevjegyek', function (Blueprint $table) {

$table->string('mobil', 13)->change(); // Visszaállítjuk az eredeti hosszra

});

}

}

3. Migráció Lefuttatása

Futtasd a migrációt a következő paranccsal:

php artisan migrate

Ezután próbáld meg újra feltölteni az adatot, és ellenőrizd, hogy sikerül-e.

2. storage:link Parancs Futtatása

Ha még nem futtattad, akkor készíts egy szimbolikus linket, hogy a storage mappában tárolt képek elérhetők legyenek a böngészőből:

php artisan storage:link

### Állítsunk elő egy végpontot, amit bármi más technológiával ki tudunk olvasni:

először is egészítsük ki a nevjegyekcontrollert:

public function jsonEndpoint()

{

// Az összes névjegy lekérdezése

$nevjegyek = Nevjegy::all();

// JSON válasz küldése

return response()->json($nevjegyek);

}

Útvonal hozzáadása:

Route::get('/endpoint', [NevjegyController::class, 'jsonEndpoint']);