Laravel

Tartalom

[Laravel 1](#_Toc181220308)

[1. Miért Laravel? 3](#_Toc181220309)

[2. Alapfogalmak Bemutatása 3](#_Toc181220310)

[2.1 MVC – modell nézet vezérlő 4](#_Toc181220311)

[2.1.1. Model – Az Adatok és a Logika Kezelése 5](#_Toc181220312)

[2.1.2. View – Az Adatok Megjelenítése 6](#_Toc181220313)

[2.1.3. Controller – A Közvetítő Réteg 6](#_Toc181220314)

[2.2. Mi az az artisan? 6](#_Toc181220315)

[2.3. Mit jelent a blade? 7](#_Toc181220316)

[2.4. Mi az az X-SLOT? 8](#_Toc181220317)

[2.5. Mi az az Eloquent model? 10](#_Toc181220318)

[3. PHP keretrendszerek 12](#_Toc181220319)

[3.1. CakePHP 12](#_Toc181220320)

[3.2. CodeIgniter 12](#_Toc181220321)

[3.3. Laravel 12](#_Toc181220322)

[3.4. Symfony 13](#_Toc181220323)

[3.5. Zend Framework (ma Laminas) 13](#_Toc181220324)

[4. Routing (Útvonalak): A web.php fájl szerepe 14](#_Toc181220325)

[4.1. Egyszerű Route-ok és Nézetek 14](#_Toc181220326)

[4.2. Controller Létrehozása és Adatok Megjelenítése a Nézetben 15](#_Toc181220327)

[4.3. Egyszerű Modell és Alapvető CRUD Műveletek 16](#_Toc181220328)

[5. Egy Laravel Alkalmazás Létrehozása 18](#_Toc181220329)

[5.1. PHP és a Laravel Telepítő telepítése 18](#_Toc181220330)

[5.2. Kezdeti Konfiguráció 19](#_Toc181220331)

[5.3. Környezeti Alapú Konfiguráció 22](#_Toc181220332)

[5.4. Adatbázisok és Migrációk 22](#_Toc181220333)

[5.4.1. Nevjegy Model Létrehozása 23](#_Toc181220334)

[5.4.2. Vezérlő Létrehozása a Névjegyek Kezelésére 23](#_Toc181220335)

[5.4.3. index Metódus az Összes Névjegy Listázására 24](#_Toc181220336)

[5.4.4. Útvonal Hozzáadása a web.php Fájlban 24](#_Toc181220337)

[5.4.5. Nézet Létrehozása az Adatok Megjelenítéséhez 24](#_Toc181220338)

[5.4.6. Gyorsítótár Törlése és Szerver Újraindítása 25](#_Toc181220339)

[5.5. Admin Middleware Létrehozása 29](#_Toc181220340)

[5.5.1. Útvonalak Beállítása az Admin Műveletekhez 30](#_Toc181220341)

[5.5.2. Nézetek Létrehozása: index, create, edit, és delete 31](#_Toc181220342)

[5.5.3. Útvonalak hozzáadása: 34](#_Toc181220343)

[5.5.4. NevjegyController CRUD Műveletek Készítése 34](#_Toc181220344)

[5.5.5. Timestamps Kezelésének Kikapcsolása a Nevjegy Modellben 39](#_Toc181220345)

[5.6. Az Adatbázis Módosítása 40](#_Toc181220346)

1. 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.

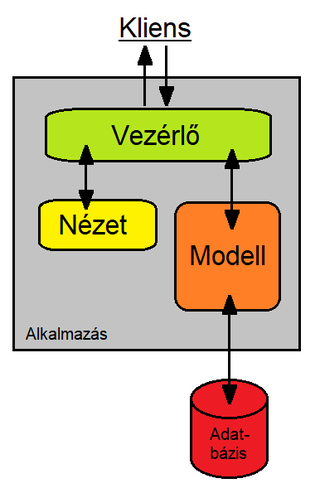
**Egy Közösségi Keretrendszer**

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.

# Alapfogalmak Bemutatása

Biztos vagyok benne, hogy kristálytisztán emlékeztek arra, hogy mit is jelent az MVC rövidítés, hiszen az angularJs kapcsán tanultuk. Nem, nem azt, hogy Mert Van itt egy Cica…

## MVC – modell nézet vezérlő

****[**https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91**](https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91)

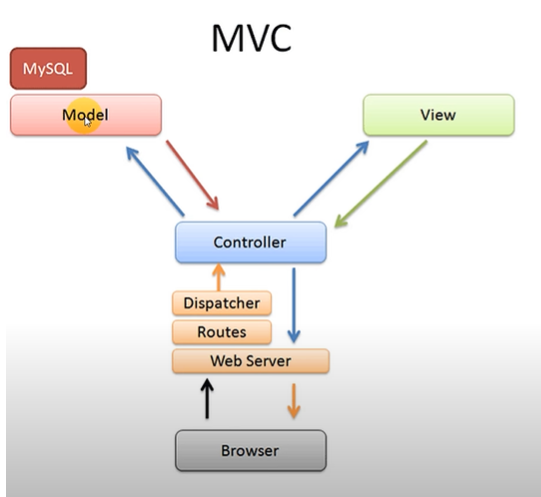
Az információt az adatot jelenti, illetve az adatkezelés szabályait tartalmazza

felhasználói felület

a controller össze kapcsolja a view-t a nézetet és a modellt, ami a felhasználóval is tartja a kapcsolatot, vagyis a usertől érkező akciók, billentyű leütések, illetve egér kattintások, ezeket feldolgozza és lekérdezéseket továbbít, a modellhez, és amit a modelltől vissza kap, azt továbbítja a viewnek, azaz a megjelenítésnek.

A böngésző két irányú kommunikációt folytat a szerverünkkel, egyrészt küld kéréseket, másrészt kap válaszokat. Elküldi a kérést a webszervernek, az átmegy bizonyos részeken és az ún controllerek, controller osztályok azok, amelyek megkapják ezeket a kéréséket. A kontroller azt csinálja, hogy egyrészt kapcsolatba lép az adatbázissal, ezt a modell osztályon keresztül teszi meg, és a modell osztályban vannak azok a függvények, amelyek lekérdezik az adatbázisból az adatokat, de semmi mással nem foglalkozik csak adatlekérdezéssel. Lekérdezi, írja, olvassa, updateli, törli az adatokat, tehát csak az adatbázissal foglalkozik.

Lekéri az adatokat a controller az adatbázisból, majd utána beolvassa a nézetet, az adatokat elhelyezi abban az állományban, amit a nézet létrehozott neki, majd ezután küldi a webszervernek, az pedig a böngészőnek vissza küldi az elkészült dolgokat. Tehát a controller az olyan, mint egy karmester a zenekarban, ő irányítja az adatbázis, ő irányítja a megjelenítést. A program kódot nagyon könnyű ez által karban tartani, vagy akár tovább fejleszteni, ha így kerül felépítésre egy rendszer. Ugyanis ha valamit szeretnénk változtatni az adatkezelésen, nem kell az egész kódot átböngésznünk, csak a modellel kell foglalkoznunk. Ha a nézeten a view-en szeretnénk változtatni akkor csak a nézettel kell foglalkoznunk, nem kell az egész kódot újra reorganizálni, átdolgozni ahhoz, hogy megváltoztassuk az alkalmazásunkat, és ez ugyanígy igaz a controllerre is.



Az **MVC (Model-View-Controller)** egy népszerű szoftverarchitektúra, amelyet a Laravel és más keretrendszerek használnak. Az MVC célja, hogy a kód struktúráját rendezetten és jól kezelhetően tartsa, azáltal, hogy a kódot három különálló rétegbe szervezi: **Model**, **View**, és **Controller**. Lássuk, hogyan működnek ezek együtt:

### Model – Az Adatok és a Logika Kezelése

A **Model** felelős az alkalmazás adatainak kezeléséért. Ez magába foglalja az adatbázishoz való kapcsolódást, az adatok lekérdezését, mentését és frissítését. Laravelben az Eloquent ORM-et (Object-Relational Mapping) használjuk, amely egyszerűsíti a különböző adatbázisműveletek végrehajtását.

* Példa: Képzeljük el, hogy van egy User model, amely tartalmazza a felhasználói adatokat (pl. name, email).
* A model lehetővé teszi, hogy a User::all() vagy User::find(1) parancsokkal könnyen lehessen adatokat lekérdezni, és Laravel automatikusan az users adatbázistáblához kapcsolódik.

### View – Az Adatok Megjelenítése

A **View** az, ami a felhasználó számára látható – itt jelenik meg az adat vizuálisan, legyen szó egy egyszerű HTML oldalról vagy egy összetettebb felhasználói felületről. Laravelben a View réteget általában Blade sablonfájlok formájában hozzuk létre a resources/views mappában.

* A View fogadja a controllertől kapott adatokat, majd megjeleníti azokat a felhasználónak.
* Példa: A welcome.blade.php fájl lehet egy egyszerű kezdőoldal, ami megjeleníti a felhasználók nevét.

### Controller – A Közvetítő Réteg

A **Controller** az összekötő kapocs a model és a view között. A controller fogadja a felhasználói kéréseket, meghatározza, milyen adatokat kell előhívni a modelből, majd továbbítja ezeket a nézet felé, hogy a megjelenítés megtörténhessen.

* Példa: A UserController nevű controller tartalmazhat olyan metódusokat, mint index() vagy show($id), amelyek a felhasználók listázását vagy egy adott felhasználó adatainak megjelenítését végzik.
* Egy tipikus folyamat:
  + A felhasználó az /users oldalra lép.
  + A UserController index() metódusa megkapja a kérést, lehívja az összes felhasználói adatot a User modelből, majd elküldi ezeket a users/index.blade.php nézetnek.

**Hogyan Működik Együtt Az MVC?**

* A felhasználó egy URL-t (például /users) kér le az alkalmazásból.
* A Laravel route-ok alapján meghívja a megfelelő controllert (pl. UserController).
* A controller meghívja a modellt, hogy hozzáférjen a szükséges adatokhoz.
* Az adatokkal a controller betölti a megfelelő nézetet, és a view-n keresztül megjeleníti azokat a felhasználónak.

Az MVC megközelítésnek köszönhetően a különböző rétegek (adatkezelés, üzleti logika, megjelenítés) elkülönülnek, így könnyebb a kód olvasása, karbantartása és tesztelése.

## Mi az az artisan?

**Artisan** a Laravel beépített parancssori felülete, amely megkönnyíti a különféle fejlesztési feladatokat. Ezzel a CLI eszközzel gyorsan generálhatunk különböző fájlokat (mint például vezérlők, modellek, migrációk), kezelhetjük az adatbázis-migrációkat, és számos más feladatot végezhetünk el egyetlen parancs kiadásával.

**Néhány fontos Artisan parancs**

1. **php artisan list**  
   Felsorolja az összes elérhető Artisan parancsot.
2. **php artisan make:controller ControllerNeve**  
   Létrehoz egy új controllert az app/Http/Controllers könyvtárban.
3. **php artisan make:model ModelNeve**  
   Létrehoz egy modellt az app/Models könyvtárban. Az -m opció hozzáadásával egy kapcsolódó migrációs fájlt is készít.
4. **php artisan migrate**  
   Futtatja az adatbázis-migrációkat, létrehozva az adatbázis-táblákat és azok oszlopait.
5. **php artisan serve**  
   Elindít egy helyi fejlesztői szervert, általában a http://127.0.0.1:8000 címen.
6. **php artisan route:list**  
   Megjeleníti az összes elérhető útvonalat, megmutatva azok metódusait, URL-jeit és a hozzájuk tartozó vezérlőket.

**Miért hasznos az Artisan?**

Az Artisan parancsok jelentősen felgyorsítják a fejlesztést, hiszen automatizálják az ismétlődő feladatokat, és segítenek a projekt struktúrájának gyors létrehozásában.

## Mit jelent a blade?

A **Blade** a Laravel beépített sablonmotorja, amelyet a nézetek (views) dinamikus tartalommal való ellátására használnak. A Blade használatával egyszerűen keverhetünk PHP kódot HTML-be, és hatékonyan hozhatunk létre újrahasznosítható sablonokat a nézetekhez.

**A Blade főbb jellemzői**

1. **Egyszerű szintaxis**

A Blade egyedi szintaxissal rendelkezik a változók megjelenítésére ({{ }}) és a vezérlési szerkezetek, mint az if és for ciklusok használatára.

<!-- Változó kiírása -->

<h1>{{ $name }}</h1>

<!-- Feltételes kiíratás -->

@if($age > 18)

<p>Felnőtt</p>

@else

<p>Kiskorú</p>

@endif

1. **Kiterjesztés és komponensek**

A Blade lehetővé teszi, hogy egy alap sablont kiterjesszünk más nézetekben, például létrehozhatunk egy layouts/app.blade.php fájlt, amit több oldalon is használhatunk. Ezzel az alapértelmezett elemeket, mint a fejléc és a lábléc, egy helyen kezelhetjük.

<!-- resources/views/layouts/app.blade.php -->

<html>

<body>

<header>@yield('header')</header>

<main>@yield('content')</main>

<footer>@yield('footer')</footer>

</body>

</html>

<!-- A fő tartalmat adó oldal -->

@extends('layouts.app')

@section('content')

<p>Főoldali tartalom</p>

@endsection

1. **Blade komponensek és slotok**

Komponensekkel újrahasznosítható elemeket készíthetünk, például gombokat, kártyákat, amiket különböző nézetekben felhasználhatunk.

1. **PHP kód beszúrása**

Bár a Blade saját szintaxist használ, lehetőséget ad arra is, hogy közvetlenül PHP kódot írjunk a @php ... @endphp blokkal.

1. **Csatlakoztatott erőforrások**

Blade lehetőséget biztosít különféle adatkezelési funkciók beépítésére, például adatokat adhatunk át, formázhatjuk őket, és könnyen integrálhatunk JavaScript-et és CSS-t.

A Blade leegyszerűsíti és rendszerezi a nézetek létrehozását, különösen az újrahasznosíthatóság és a tiszta szintaxis miatt. Könnyű kiterjeszteni más nézeteket és rendszerezni a weboldal különböző részeit, így a karbantartás és a fejlesztés is átláthatóbbá válik.

## Mi az az X-SLOT?

A **x-slot** a Blade komponensek része Laravelben, amelyet arra használnak, hogy egy Blade-komponens adott részéhez (slot) tartalmat adjanak meg. A komponensek és slotok segítségével újrahasznosítható felépítéseket hozhatunk létre, például kártyákat, modális ablakokat vagy táblázatokat, ahol minden komponens résztestre szabható.

**Hogyan működik az x-slot?**

Amikor egy komponenst hozunk létre, különféle slot elemeket definiálhatunk benne, hogy lehetővé tegyük a komponens különböző részeinek testreszabását. Az x-slot segítségével megadhatjuk, melyik részbe szeretnénk tartalmat helyezni.

**Példa: Komponens és x-slot használata**

1. **Komponens létrehozása**

Hozz létre egy Blade-komponenst, amely például egy kártyát reprezentál. Ez lehet például resources/views/components/card.blade.php.

<!-- resources/views/components/card.blade.php -->

<div class="card">

<div class="card-header">

{{ $header }}

</div>

<div class="card-body">

{{ $slot }}

</div>

<div class="card-footer">

{{ $footer }}

</div>

</div>

Ebben a példában három részt definiálunk: **header**, **body** (a fő tartalom, ami slot néven érhető el) és **footer**.

1. **A komponens használata egy nézetben**

Most, hogy megadtuk a card komponenst, beilleszthetjük egy nézetbe, például home.blade.php, és az x-slot segítségével adhatunk meg külön tartalmat az egyes részekhez:

<!-- resources/views/home.blade.php -->

<x-card>

<x-slot name="header">

Ez a fejléc

</x-slot>

Itt található a kártya fő tartalma.

<x-slot name="footer">

Ez a lábléc

</x-slot>

</x-card>

* + **<x-slot name="header">**: Ezzel a header slotba helyezzük a tartalmat.
  + **$slot**: Maga a fő tartalom kerül ide, amit az x-card komponens hívása után írunk.
  + **<x-slot name="footer">**: Ezzel a footer slotba helyezzük a tartalmat.

Az x-slot lehetővé teszi, hogy egyetlen komponensen belül több részt különbözőképpen töltsünk fel tartalommal. Ezáltal egyszerűbbé és átláthatóbbá válik az újrahasznosítható komponensek használata, és sokkal rugalmasabb felépítést tesz lehetővé.

## Mi az az Eloquent model?

Az **Eloquent model** a Laravel ORM (Object-Relational Mapper) eszköze, amely az adatbázis-kezelést teszi egyszerűvé és hatékonyabbá a PHP alapú Laravel alkalmazásokban. Az Eloquent modellek automatikusan összekötik az adatbázis táblákat a PHP osztályokkal, és lehetővé teszik, hogy az adatokat objektumokként kezeljük az alkalmazáson belül.

**Hogyan működik az Eloquent model?**

1. **Modellek és táblák kapcsolata**

Egy Eloquent modell általában egy adatbázis-táblát reprezentál. Például, ha van egy users nevű táblánk az adatbázisban, akkor egy User modell hozható létre, amely automatikusan az users táblára hivatkozik.

1. **CRUD műveletek egyszerűsítése**

Az Eloquent segítségével egyszerűen végezhetünk alapvető adatbázis-műveleteket, mint például létrehozás (Create), lekérdezés (Read), frissítés (Update) és törlés (Delete), amelyek az Eloquent röviden csak CRUD műveletekként emleget.

1. **Kapcsolatok kezelése**

Az Eloquent lehetőséget nyújt különböző relációk (kapcsolatok) kezelésére is az adatbázis-táblák között. Így megadhatjuk például, hogy egy User modellhez több Post modell tartozik (egy- a- többhöz kapcsolat), vagy egy Post modellhez több Tag kapcsolódik (több- a- többhöz kapcsolat).

**Példa egy egyszerű Eloquent modell létrehozására**

1. **Modell létrehozása**

A modell létrehozásához a következő parancsot használhatjuk:

php artisan make:model Post

Ez a parancs létrehozza a Post modellt az app/Models könyvtárban.

1. **Egyszerű lekérdezések és CRUD műveletek**

A Post modell automatikusan az posts táblára vonatkozik. Ha például hozzáadunk egy új bejegyzést, vagy lekérdezzük az összes bejegyzést, az alábbi kódot írhatjuk:

use App\Models\Post;

// Minden bejegyzés lekérdezése

$posts = Post::all();

// Új bejegyzés létrehozása

$post = new Post;

$post->title = 'Az első bejegyzés';

$post->content = 'Ez a tartalom.';

$post->save();

1. **Relációk definiálása**

Ha egy Post modellhez hozzárendeljük, hogy egy User modell tulajdona (egy- a- többhöz kapcsolat), akkor azt az alábbiak szerint definiálhatjuk:

class Post extends Model

{

public function user()

{

return $this->belongsTo(User::class);

}

}

* **Egyszerűsített adatkezelés**: Az adatbázis-műveletek egyszerűek és olvashatóak.
* **Kapcsolatok könnyű kezelése**: Egy- a- egyhez, egy- a- többhöz, és több- a- többhöz kapcsolatok kezelése egyszerű.
* **Beépített eszközök**: Pagináció, rendezés, keresés és feltételes lekérdezések támogatása.

Az Eloquent modell nagyban segíti a Laravel fejlesztőket az adatbázissal való munkában, mivel az SQL lekérdezések helyett objektum-orientáltan dolgozhatnak, így az adatok kezelése olvashatóbbá és könnyebben karbantarthatóvá válik.

# PHP keretrendszerek

pl: CakePhp, Codelgniter, Laravel, Symfony, Zend Framework

## CakePHP

* **Előnyök**:
  + **Gyors fejlesztés**: Beépített CRUD (Create, Read, Update, Delete) generátor, amely gyorsítja az alapvető alkalmazáskomponensek létrehozását.
  + **Egyszerű konfiguráció**: Nincs szükség sok előzetes konfigurációra.
  + **Beépített biztonsági eszközök**: Alapértelmezetten támogatott az SQL injection és XSS támadások elleni védelem.
* **Hátrányok**:
  + **Kevésbé rugalmas**: A szoftverstruktúra és a mappák elrendezése szigorúan szabályozott, ami nehezítheti az egyedi megoldások beépítését.
  + **Dokumentáció és közösség mérete**: Bár aktív közössége van, kisebb és kevesebb erőforrással rendelkezik, mint a Laravel vagy Symfony. Mivel kijött az újabb verziója, és nem találtam megoldást a régi feladataim tovább fejlesztésére én el is engedtem végleg, de egy 3-as verziót még tudok mutatni, utoljára két éve csináltunk végig egy feladatot vele.
* **Elterjedtség**: Mérsékelt népszerűséggel bír, leginkább kis- és középvállalati projektekhez használják.
* **Népszerűség**: Csökkent az utóbbi években, de továbbra is használt.

## CodeIgniter

* **Előnyök**:
  + **Könnyű és gyors**: Nagyon gyors és könnyű, ideális egyszerű, nagy teljesítményű alkalmazásokhoz.
  + **Kisebb tanulási görbe**: Könnyen megtanulható és használható, minimális függőségekkel.
  + **Rugalmas struktúra**: Nincsenek szigorú szabályok az alkalmazás felépítésére, így többféleképpen is kialakítható.
* **Hátrányok**:
  + **Hiányos beépített funkciók**: Kevesebb beépített eszközzel rendelkezik, mint más keretrendszerek, például ORM és automatikus validáció hiányzik.
  + **Korlátozott közösségi támogatás**: A közössége kisebb, mint a Laravel vagy Symfony esetében.
* **Elterjedtség**: Használt, különösen olyan fejlesztők körében, akik egyszerű és gyors megoldást keresnek.
* **Népszerűség**: Az elmúlt években csökkent, de még mindig népszerű az egyszerűbb projektekhez.

## Laravel

* **Előnyök**:
  + **Gazdag funkcionalitás**: Számos beépített funkcióval rendelkezik, például Eloquent ORM, automatikus validáció, kényelmes routing, valamint Blade sablonmotor.
  + **Erős közösség és dokumentáció**: Laravel rendelkezik az egyik legnagyobb és legaktívabb közösséggel, rengeteg dokumentáció és oktatási anyag áll rendelkezésre.
  + **Könnyű integráció modern technológiákkal**: Laravel könnyen használható Vue.js-sel, Reacttal és modern front-end eszközökkel.
  + **Erős támogatás a RESTful és API fejlesztéshez**.
* **Hátrányok**:
  + **Teljesítményigény**: Laravel nagyobb szerver-erőforrásokat igényel, mint más könnyebb keretrendszerek, mint a CodeIgniter.
  + **Nagyobb tanulási görbe kezdőknek**: Az újonnan tanulók számára eleinte bonyolult lehet, mivel rengeteg funkcióval rendelkezik.
* **Elterjedtség**: A legnépszerűbb PHP keretrendszer, mind a kis, mind a nagy projektekhez, valamint az egyéni fejlesztők és nagyobb fejlesztői csapatok körében is elterjedt.
* **Népszerűség**: Jelenleg a legnépszerűbb PHP keretrendszer a piacon, ezért ezt választottam, és Nektek is ezt fogom megmutatni.

## Symfony

* **Előnyök**:
  + **Modularitás és újrahasználhatóság**: A keretrendszer rendkívül moduláris, számos különálló csomagból áll, amelyeket külön-külön is lehet használni.
  + **Erős közösség és nagyvállalati támogatás**: Nagyvállalatok által is használt, valamint az egyik legszélesebb körben támogatott PHP keretrendszer.
  + **Szabványok követése**: Symfony szigorúan betartja az ipari szabványokat, ami könnyebbé teszi a kód újrahasznosítását más projektekben.
* **Hátrányok**:
  + **Komplexitás**: Nagyobb tanulási görbe és bonyolultabb beállítási folyamat, különösen kezdők számára.
  + **Magasabb erőforrásigény**: Nagyobb erőforrásigénnyel bír, ami nagyobb terhelés mellett csökkentheti a teljesítményt.
* **Elterjedtség**: Különösen nagyvállalati alkalmazásokban és olyan projektekben használják, ahol fontos a szabványok betartása.
* **Népszerűség**: Nagyon elterjedt a komplexebb, hosszabb távú projekteknél.

## Zend Framework (ma Laminas)

* **Előnyök**:
  + **Stabil és robusztus**: Nagyvállalati szintű alkalmazások fejlesztésére készült, különösen erős a teljesítmény és biztonság terén.
  + **Moduláris kialakítás**: Minden komponens külön csomagként érhető el, így szükség szerint használhatók.
  + **Nagyvállalati támogatás**: Laminasként folytatódott a fejlesztése, és vállalati környezetben népszerű.
* **Hátrányok**:
  + **Magas tanulási görbe**: Az összetettség miatt lassabban tanulható meg, és sok beállítást igényel.
  + **Kisebb közösség**: A Zend Framework neve alatt már nem folytatódik a fejlesztés, helyette a Laminas projekt létezik, így a régi Zend közösség szétszóródott.
* **Elterjedtség**: Több nagyvállalat is használja, főként olyan helyeken, ahol stabil és skálázható megoldásra van szükség.
* **Népszerűség**: A Laminasra való áttérés óta a népszerűsége csökkent, de továbbra is elismert nagyvállalati keretrendszer.

**Általános Elterjedtség és Népszerűség**

A jelenlegi piaci trendek alapján **Laravel** a legnépszerűbb és legelterjedtebb PHP keretrendszer, köszönhetően a könnyű tanulhatóságának, az erős közösségi támogatásnak, valamint a széles körű funkcióknak. A **Symfony** szintén elterjedt, különösen nagyvállalati környezetben és komplexebb projektekben. A **CodeIgniter** és **CakePHP** kisebb projektekhez és egyszerűbb alkalmazásokhoz használatosak, míg a **Laminas** a nagyvállalati megoldások között található meg, de népszerűsége mérséklődött.

# ****Routing (Útvonalak): A**** web.php ****fájl szerepe****

A Laravel alkalmazásban a **route-ok** segítségével tudjuk meghatározni, hogy a beérkező kérések milyen műveletet végezzenek, és hogy a felhasználókat hova irányítsák az alkalmazásban. A route-ok meghatározása a **routes/web.php** fájlban történik.

A Laravel a különböző route-okat az URL és a HTTP-metódus (például GET, POST) alapján irányítja a megfelelő kontrollerhez vagy funkcióhoz. Íme egy példa az egyszerű útvonal meghatározására:

// Egyszerű útvonal egy üzenet megjelenítéséhez

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

return 'Hello, World!';

});

* **Magyarázat**: Az előző példában a Route::get egy GET kérést kezel. Amikor a felhasználó meglátogatja a /hello URL-t, a rendszer egyszerűen visszaadja a Hello, World! üzenetet.

## ****Egyszerű Route-ok és Nézetek****

A Laravelben nézetekkel (views) jelenítjük meg a felhasználónak az adatokat és az alkalmazás felületét.

**a) Route::get('/hello', function () { return 'Hello, World!'; });**

Ez egy egyszerű útvonal, amely egy üzenetet jelenít meg, ha a /hello URL-t meglátogatják. Laravel automatikusan kezeli az útvonalakat a routes/web.php fájlban, így ez az útvonal automatikusan elérhető lesz az alkalmazásban.

**b) Egyszerű nézet létrehozása**

Hozzunk létre egy új nézetet, ami egy statikus tartalmat jelenít meg.

1. **Lépjünk a resources/views könyvtárba**, és hozzunk létre egy home.blade.php fájlt.
2. A fájlba írjuk be a következőt:

<!-- resources/views/home.blade.php -->

<h1>Welcome to the Home Page!</h1>

<p>This is a simple static view.</p>

1. Ezt a nézetet egy route segítségével érhetjük el:

// A web.php fájlban

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

return view('home');

});

* + **Magyarázat**: A view('home') kifejezés a resources/views/home.blade.php nézetet tölti be.

## ****Controller Létrehozása és Adatok Megjelenítése a Nézetben****

A controller a kéréseket kezeli, adatokat küld a nézetnek, és összeköti a modellek és nézetek működését.

**a) Controller létrehozása**

1. Hozz létre egy kontrollert az Artisan CLI segítségével:

php artisan make:controller HomeController

* + Ezzel a parancssal egy új HomeController.php fájl jön létre az app/Http/Controllers könyvtárban.

1. Nyisd meg a HomeController fájlt, és add hozzá az index metódust:

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

public function index()

{

$message = "Hello from HomeController!";

return view('home', compact('message'));

}

}

* + A fenti példában az index metódus egy message változót küld a home nézetnek.

**b) Adatok megjelenítése a nézetben**

1. A home.blade.php fájlban jelenítsük meg az adatot:

<!-- resources/views/home.blade.php -->

<h1>{{ $message }}</h1>

1. Készítsünk egy route-ot a controller hívásához:

Route::get('/home', [HomeController::class, 'index']);

* + **Magyarázat**: A compact('message') használatával a $message változót átadjuk a nézetnek, amely a {{ $message }} segítségével megjeleníthető.

## ****Egyszerű Modell és Alapvető CRUD Műveletek****

Az Eloquent ORM segítségével a Laravel egyszerűsíti az adatbázis műveleteket, mint például az adatok létrehozását, frissítését, törlését, és megjelenítését.

**a) Modell létrehozása**

1. Hozz létre egy modellt az artisan CLI-vel:

php artisan make:model Product

* + Ez létrehoz egy Product.php modellt az app/Models mappában.

1. Hozz létre egy egyszerű adatbázistáblát ehhez a modellhez migráció segítségével:

php artisan make:migration create\_products\_table --create=products

1. A create\_products\_table migrációban adj hozzá néhány mezőt a táblához:

public function up()

{

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

$table->id();

$table->string('name');

$table->decimal('price', 8, 2);

$table->timestamps();

});

}

1. Futtasd a migrációt:

php artisan migrate

**b) Adatok Listázása**

1. Készíts egy controllert az adatbázisadatok lekéréséhez:

php artisan make:controller ProductController

1. Az index metódusban kérd le az összes adatot:

namespace App\Http\Controllers;

use App\Models\Product;

use Illuminate\Http\Request;

class ProductController extends Controller

{

public function index()

{

$products = Product::all();

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

}

}

1. Hozz létre egy index.blade.php nézetet a termékek listázásához a resources/views/products könyvtárban:

<!-- resources/views/products/index.blade.php -->

<h1>Products List</h1>

<ul>

@foreach ($products as $product)

<li>{{ $product->name }} - ${{ $product->price }}</li>

@endforeach

</ul>

1. Adj hozzá egy route-ot a web.php fájlban:

Route::get('/products', [ProductController::class, 'index']);

**c) Adatok Létrehozása**

Hozz létre egy create metódust a ProductController-ben egy form megjelenítéséhez, és egy store metódust az adatok mentéséhez:

public function create()

{

return view('products.create');

}

public function store(Request $request)

{

$request->validate([

'name' => 'required|string|max:255',

'price' => 'required|numeric',

]);

Product::create($request->only(['name', 'price']));

return redirect()->route('products.index');

}

Hozz létre egy create.blade.php nézetet a termék létrehozásához:

<!-- resources/views/products/create.blade.php -->

<h1>Add New Product</h1>

<form action="{{ route('products.store') }}" method="POST">

@csrf

<input type="text" name="name" placeholder="Product Name">

<input type="text" name="price" placeholder="Product Price">

<button type="submit">Save Product</button>

</form>

Hozz hozzá egy route-ot a web.php-ban:

Route::get('/products/create', [ProductController::class, 'create']);

Route::post('/products', [ProductController::class, 'store'])->name('products.store');

Ezek a lépések segítenek megérteni az alapvető Laravel struktúrát, és biztosítják a szükséges alapismereteket a bonyolultabb feladatok megkezdéséhez.

1. Egy Laravel Alkalmazás Létrehozása
   1. 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.

* 1. 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.

## 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.

### 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'];

}

### ****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.

### 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'));

}

### ****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');

### ****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>

### ****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.

**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.

## ****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'));

}

}

### ****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');

});

### ****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');

});

### ****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']);