LiteNotes

Tartalom

[1. Specifikáció 2](#_Toc127733948)

[2. Készítsünk új projektet: 4](#_Toc127733949)

[3. Breeze és Jetstream 5](#_Toc127733950)

[4. Kezdő lap cseréje 7](#_Toc127733951)

[5. Title átírása 7](#_Toc127733952)

[6. Első migráció végrehajtása 7](#_Toc127733953)

[7. Útvonalak megadása 8](#_Toc127733954)

[8. Bejegyzések beolvasása 9](#_Toc127733955)

[9. Jegyzetek renderelése a böngészőben 12](#_Toc127733956)

[10. Lapozó hozzáadása 14](#_Toc127733957)

[11. Új feljegyzés hozzáadása a weboldalról 15](#_Toc127733958)

[12. Űrlapon felvitt adatok mentése az adatbázisba 17](#_Toc127733959)

[13. Bejegyzések címének linkké alakítása 20](#_Toc127733960)

[14. Egyedi azonosítók 21](#_Toc127733961)

[15. Bejegyzések szerkesztése 23](#_Toc127733962)

[16. Törlés 25](#_Toc127733963)

[17. Visszajelzés a usernek a végrehajtott művelet sikerességéről 25](#_Toc127733964)

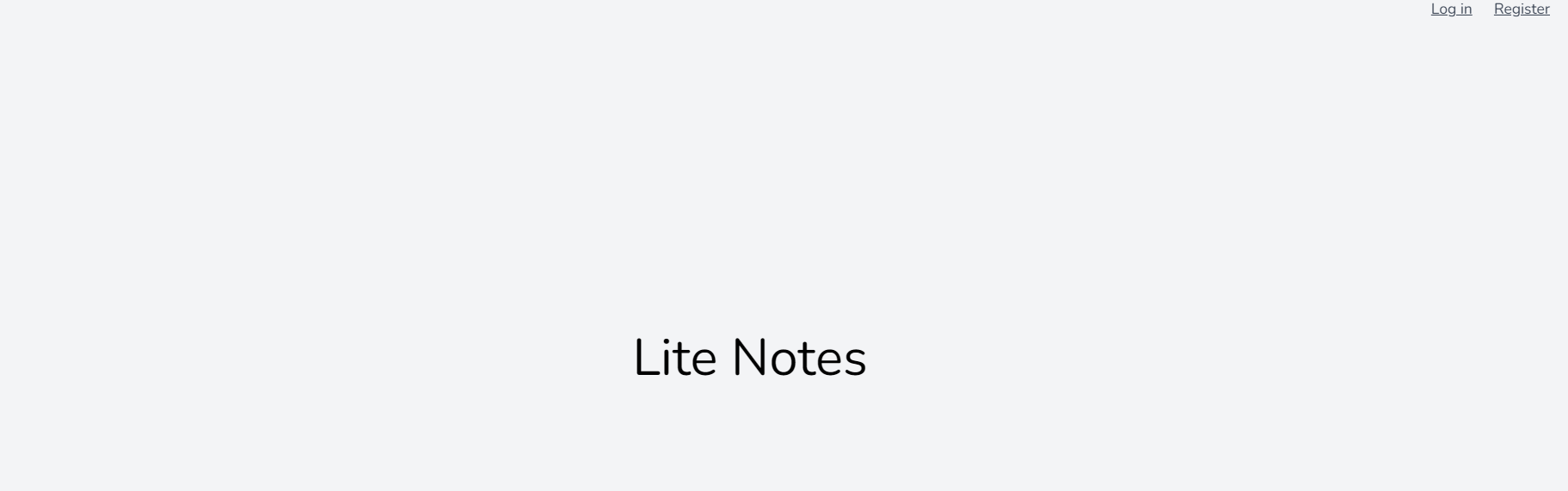
[18. Eloquent kapcsolatok 26](#_Toc127733965)

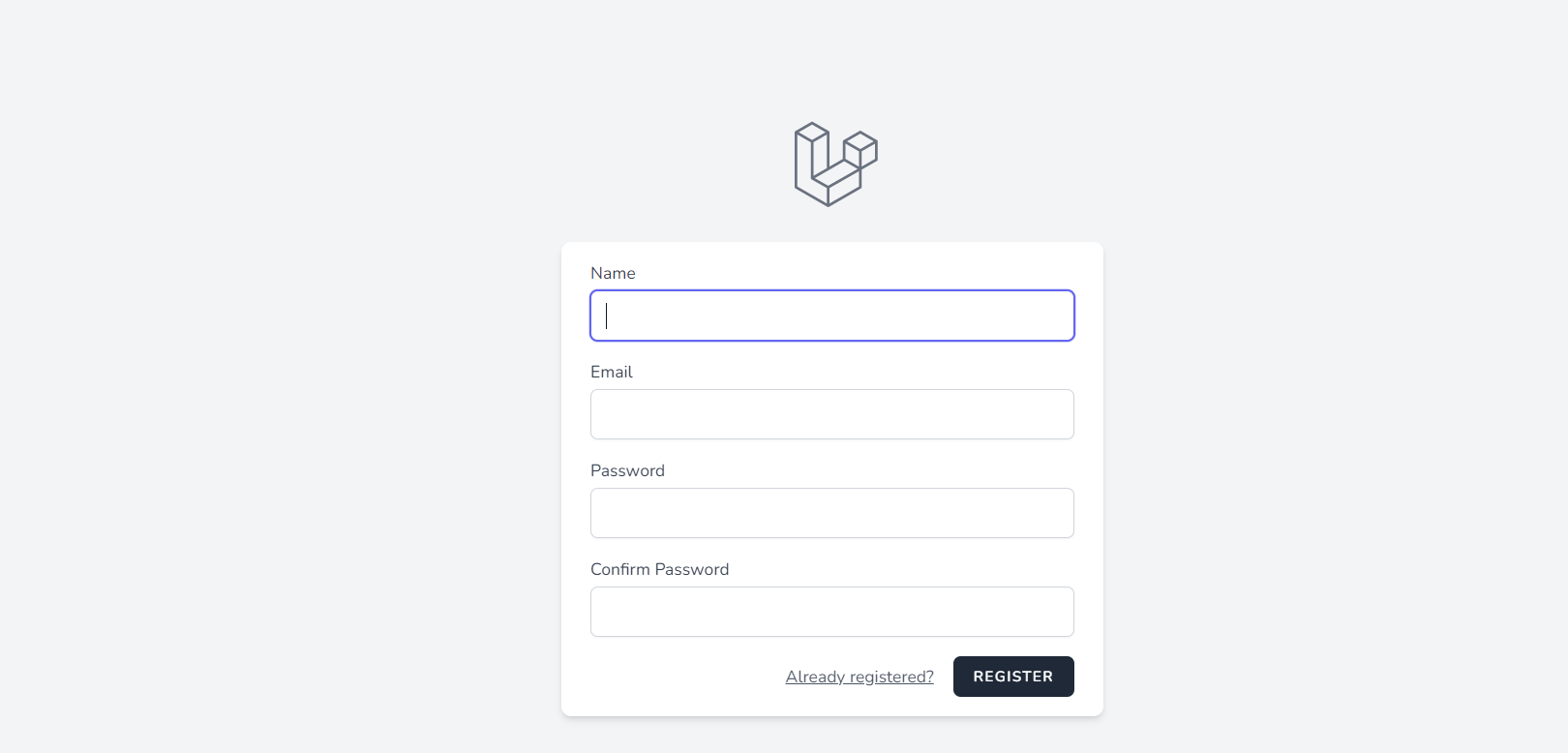
[19. Soft delete – azaz a törlés után visszaállítható legyen még a jegyzet 29](#_Toc127733966)

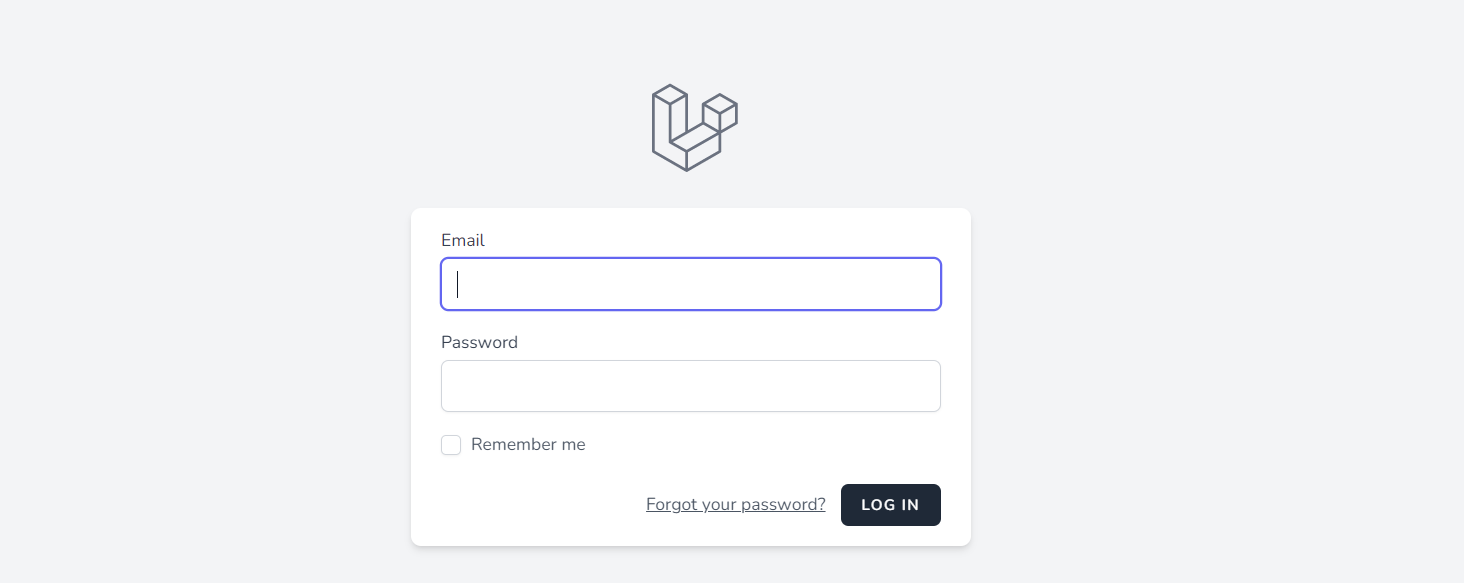
[20. Végleges törlés 36](#_Toc127733967)

# Specifikáció

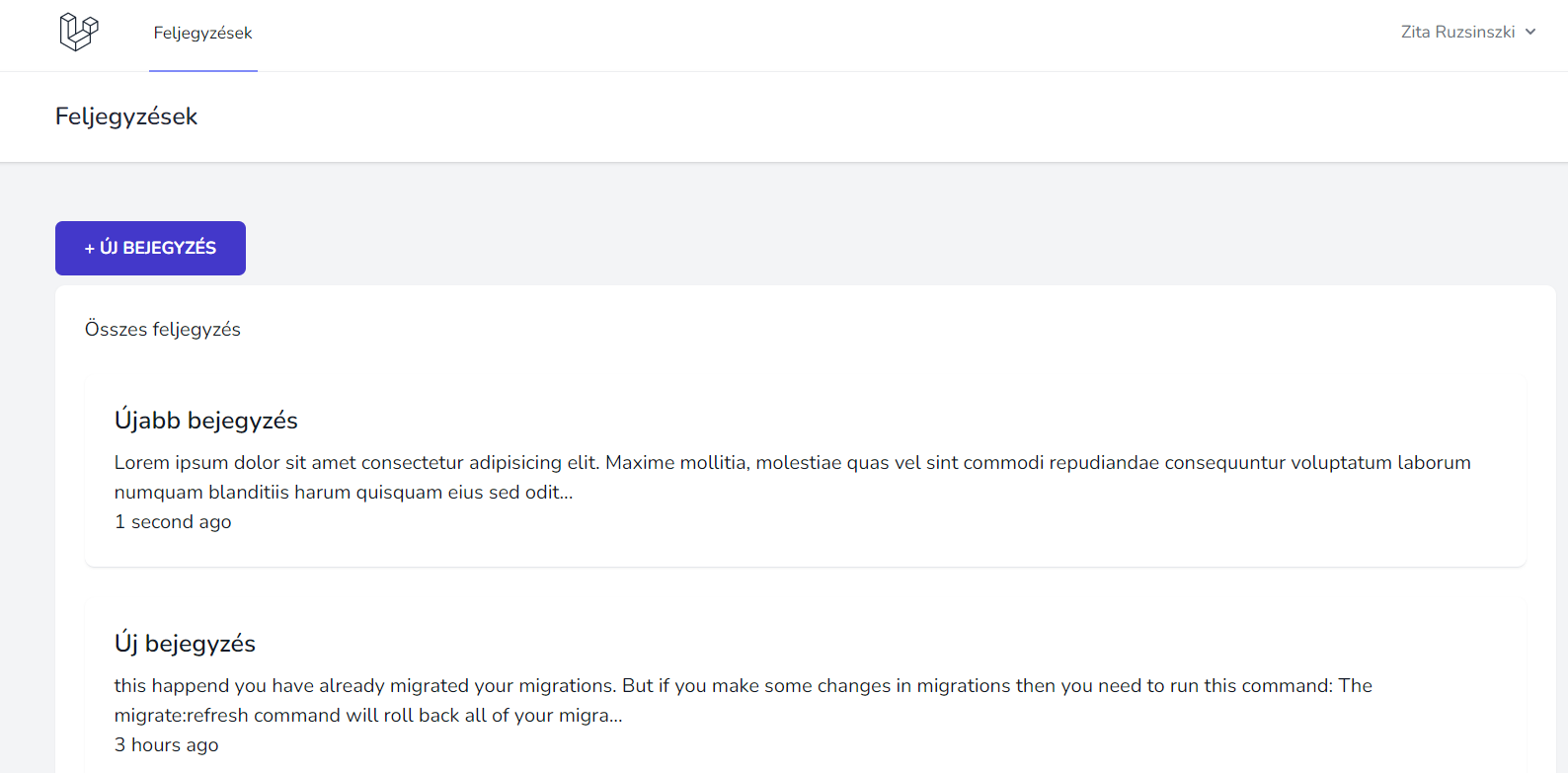
Egy egyszerű jegyzetkészítő webes alkalmazást készítünk. Az elkészítendő alkalmazás funkciói: a főoldalra egy bejelentkezés, és egy regisztrációs lehetőség menü kerül.

 Az új felhasználónak névvel, e-mail címmel és jelszóval kell regisztrálnia. Ez létrehoz egy új fiókot, és bejelentkezteti a felhasználót.

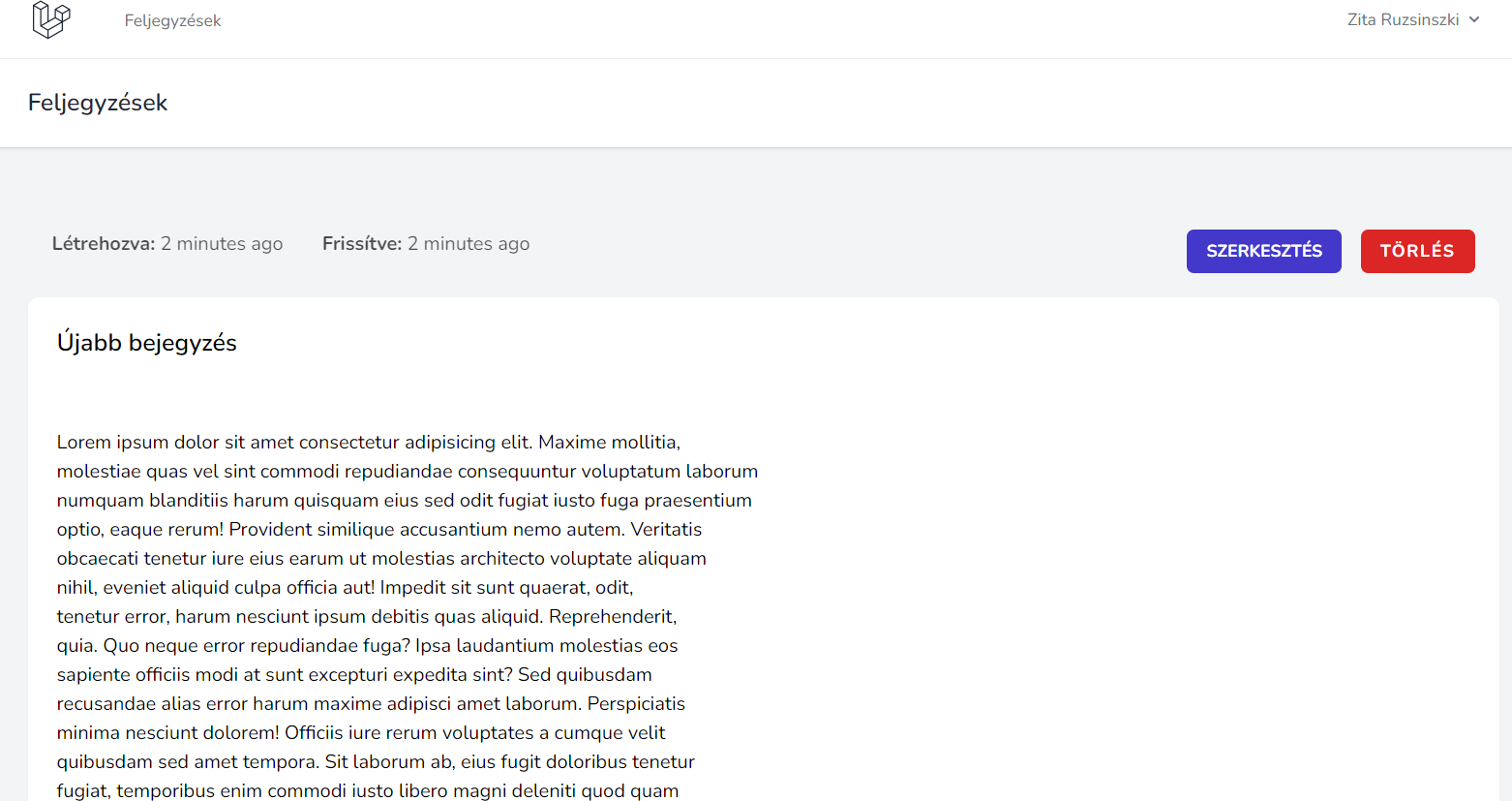
 Egy meglévő felhasználó bejelentkezhet a bejelentkezési oldalról.



Amint a felhasználó bejelentkezik, látni fogja az összes jegyzetét.



A rendszer úgy rendezi őket, hogy először a legfrissebb jegyzeteket jeleníti meg. Oldalanként öt jegyzet látható, majd alul egy oldalszámozás. Innen közvetlenül létrehozhat új jegyzetet, címet adhat, szöveget adhat hozzá. Mindkettő kötelezően kitöltendő mező lesz, tehát ha felhasználó bármelyiket üresen hagyja, a rendszer üzenetet küld. Bármely jegyzetre kattintva megtekinthetjük a teljes jegyzetet. Minden jegyzet egyedi azonosítóval rendelkezik, ami az url-ben meg is jelenik, látható lesz az is, hogy mikor készült a jegyzet, és mikor frissítették utoljára. A jegyzeteket lehet majd szerkeszteni vagy törölni is. A törölt jegyzetet még vissza lehet állítani, akkor vissza kerül a jegyzetek közé. A végleges törlést a kukából lehet majd végrehajtani, amihez a rendszer megerősítést fog kérni.



Teljes bejelentkezési folyamatot viszünk végig, regisztrációval, bejelentkezéssel, kijelentkezéssel és jelszó visszaállításával. Jegyzeteket készíthetsz, megtekintheted a listát, megtekintheted az egyes jegyzeteket, szerkesztheted őket, áthelyezheted a kukába, visszaállíthatod a törölt elemeket, és örökre törölheted a jegyzeteket. Kezdjük el felépíteni a LiteNotes alkalmazást.

# Készítsünk új projektet:

puttyon lépjünk be a code mappába és adjuk ki a laravel new litenotes parancsot, a jegyzettömböt futtassuk rendszergazdai módban, és a windows – sys32-drivers-etc mappákon belül a host fájlhoz adjuk hozzá a 192.168.56.56 litenotes útvonalat is, majd nyissuk meg a homestead mappán belül a homestead.yaml fájlt és egészítsük ki a sites-ot két sorral:

- map: litenotes.test

to: /home/vagrant/code/litenotes/public

valamint a database-hez is adjuk hozzá: litenotes\_db. Ezek után a parancssorban adjuk ki a vagrant provision parancsot, amely újra indítja a virtual boxot.

Ha a böngészőben beírjuk: litenotes.test akkor jól csináltunk mindent indulhat a munka. ☺

Állítsuk be a .env fájlt:

DB\_CONNECTION=mysql

DB\_HOST=localhost

DB\_PORT=3306

DB\_DATABASE=laraveltest\_db

DB\_USERNAME=root

DB\_PASSWORD=

Amit megvalósítunk: a regisztráció, bejelentkezés, kijelentkezés és jelszó-visszaállítás összes funkciója a hitelesítésen kívül. Sok időbe telne, hogy ezeket manuálisan megépítsük, de szerencsére a Laravel rendelkezik néhány kezdő készlettel, amelyekkel ez nagyon egyszerű és gyors.

# Breeze és Jetstream

Két lehetőség áll rendelkezésünkre: a Breeze és Jetstream. A Breeze egyszerű és minimális kiindulópontot biztosít a hitelesítéshez, míg a Jetstream fejlettebb funkciókat kínál.

A Breeze a Tailwind CSS-t használja a stílushoz, egyszerű vele dolgozni még akkor is, ha nincs előzetes ismeretünk a Tailwind CSS-ről. De ha valaki úgy dönt, hogy egyáltalán nem szeretné a Tailwind CSS -t a projektben, választhat egy olyan csomagot, mint a Laravel UI, amely a bootstrap keretrendszert használja.

A Breeze-t fogjuk használni projektünkben, mivel Marcell utálja a Bootstrapet. ☺

<https://laravel.com/docs/9.x/starter-kits> itt láthatjuk is a mintát, amit létre lehet vele hozni. Picit ha lejjebb tekerünk, akkor ott meg is találjuk a kiadandó parancsot, adjuk ki terminálban:

composer require laravel/breeze --dev

telepításkor a blade –et adjuk meg 0, utána néhány yes, majd egymás után a következő parancsokat, remélve, hogy nodejs-t már telepítettünk a gépetekre.

php artisan breeze:install

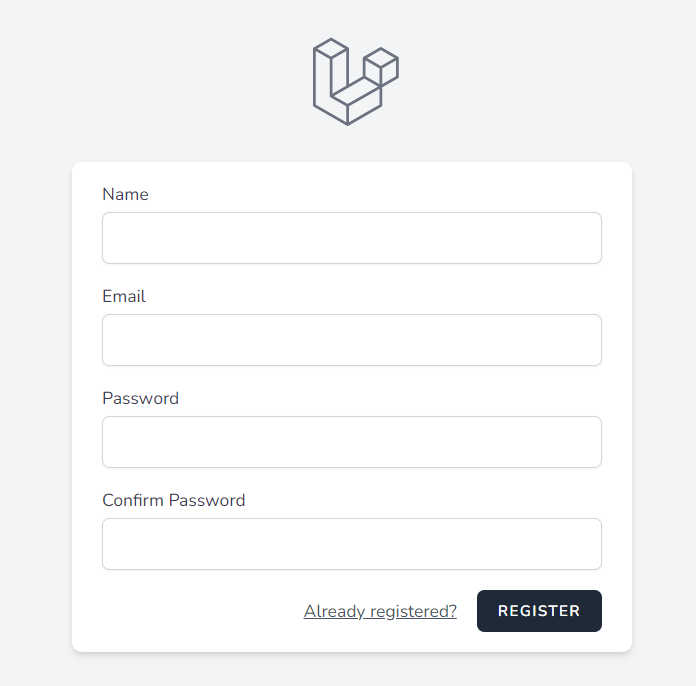
php artisan migrate

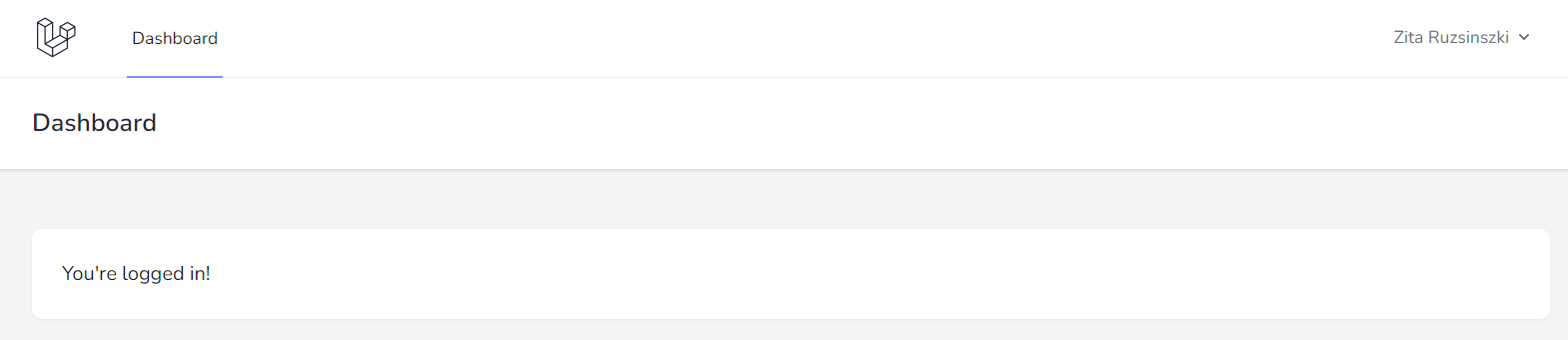
npm install

npm run dev

Az artisan sell artisan Breeze install parancs hozzáadja a hitelesítési rootokat, nézeteket, vezérlőket és egyéb szükséges erőforrásokat projektünkhöz. Az NPM install és az NPM run dev futtatására azért van szükség, mert a Breeze hozzáadott néhány CSS- t, amelyet most le kell fordítani a stílusok működéséhez. (ha nincs npm és nodejs akkor ezeket telepíteni kell)

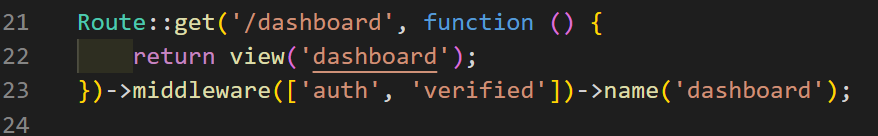
<http://localhost/laravel/public/register> nézzük meg a böngészőben az eredményt

 Remélem ezt látod, ha igen töltsd ki a regisztrációt, de először próbálj meg 8 karakternél rövidebb jelszót adni, és hibásan kitölteni az egészet.



Nyisd le a neved alatt, és ott kiválaszthatod a profilod megnézését, vagy a kilépést. Ha kilépsz vissza visz a főoldalra, ha a linkbe megadod <http://litenotes.test/login> akkor a bejelentkezéshez visz, ahol újra be tudsz jelentkezni. (Az Elfelejtett jelszó és a jelszó visszaállítása csak akkor fog működni, ha hozzáadjuk a levelezőszerver hitelesítő adatait az .nv fájlhoz )

Keressük ki a routes mappában a web.php fájlt, ahol az útvonalakat láthatjuk.

 A dashboard egy middleware-t használ, vagyis, ha kipróbáljuk, hogy beírjuk a böngészőbe: <http://litenotes.test/dashboard> (ha ki vagyunk lépve, vissza fog dobni a login oldalra először, nem enged be a rendszerbe. Ez azt jelenti, hogy ezt az útvonalat csak bejelentkezett felhasználók érhetik el.

 az összes autentikáció ebben a fájlban található, kattints az auth.php fájlra, és ott láthatod benne az összes kontrollert. Nézzük meg ezek, hol kerültek hozzáadásra a működéshez. **App -> Http -> controllers > Auth** mappában.

Következő: resources ->views -> és ezen belül 3 további összetevőt láthatunk, az auth, components, és layouts mappákat, valamint 2 blade fájl van hozzáadva. Mindezeket a nézetfájlokat komponensekkel és elrendezésekkel kombinálva hozzáadtuk.

# Kezdő lap cseréje

Cseréljük le a kezdőoldalt töröljünk le mindent: **Resources – view – welcome.blade.php**

Töröljük ki a style tagek közül az összes stílust, nem lesz rá szükség. Ha jól megnézzük a body-ban meg van adva a login és a regisztráció, azokat hagyjuk meg, az alatta lévő összes többit töröljük ki, utolsó záró div + body és html zárók maradjanak. Ha ráfrissítünk a belépő oldalra, akkor látható, hogy az összes stílus formázásunk oda lett, viszont ha a login oldalra lépünk, ott van egy kellemes formátum. Keressük meg, honnan kapja meg ezt a stílust és másoljuk át kezdő lapra. Itt találod: resources\views\auth\login.blade.php itt láthatjuk, hogy x-guest layout koponens került felhasználásra, ami a layout mappában található: resources\views\layouts\guest.blade.php itt megtaláljuk a css-re való hivatkozást, másoljuk át:

@vite(['resources/css/app.css', 'resources/js/app.js'])

a welcome.blade.php-be. Nézzük meg, azonnal látható, hogy megkapta ugyanazt a stílust.

A blade template után adjuk hozzá: <h1 class="text-5xl">Lite Notes</h1>

A public mappán belül találjuk meg a css-t, a build mappán belül. Ha egyéb css –t akarunk hozzá adni akkor itt tehetjük meg: **resources\css\app.css**

a tailwindre ha a vsc azt írja unknown rules, megoldás: (de működik e nélkül is)

<https://stackoverflow.com/questions/71217076/laravel-unknown-at-rule-tailwindcssunknownatrules>

Lényeg, hogy azt a 3 sort ne távolítsuk el, mert a breeze ezt használja, de ki is próbálhatod, mi fog történni, ha kitöröld.

# Title átírása

A welcome.blade.php-ben javítsuk ki a title-t.

<title>{{ config('app.name') }}</title>

ahhoz, hogy ne a Laravel jelenjen meg a böngésző fülön, az .env fájlban is ki kell javítani az első sorban LiteNotes-ra. (nem lehet közte szóköz)

# Első migráció végrehajtása

Most készítsük el az első modellünket a migrációval együtt. Jegyzetkészítő alkalmazást készítünk. Tehát egyértelműen szükségünk van egy jegyzettáblázatra és egy megfelelő jegyzetmodellre. A terminálon adjuk ki a következő parancsot:

php artisan make:model Note –m

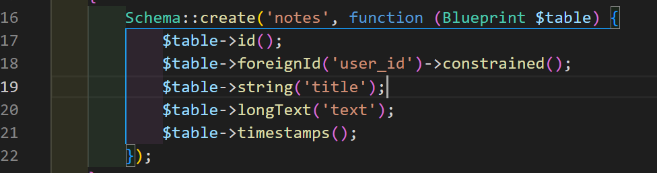
A modellnévnek nagybetűvel kell kezdődnie, és egyes számban kell lennie, ezért Note és nem notes. Ugyanez a parancs egy -m opcióval létrehozza a táblát is.

Ellenőrizzük létre jött e a fájl: **app\Models\Note.php**

Majd ellenőrizzük: database\migrations\[éppen most datum és idő]\_create\_notes\_table.php

Ezzel megvan a migráció. Már vannak előre kitöltött mezőink, mint id, és timestamp.

Adjunk hozzá oszlopokat, title, text, és szükség lesz egy idegenkulcsra, hogy tudjuk, ki hozta létre a bejegyzést.

 $table->foreignId('user\_id')->constrained();

$table->string('title');

$table->longText('text');

Most így minden jegyzet egy felhasználóhoz tartozik. Tehát a felhasználói azonosítót idegen kulcsként kell megadnunk, ami itt van, a User ID lesz az oszlop neve. Migrálhatunk is. Terminálra, és:

php artisan migrate

Nyissuk meg a böngészőben az adminert és nézzük meg létrehozta e a táblát. Ha igen, akkor készen van a Note modellünk is.

# Útvonalak megadása

Nyissuk meg a web.php fájlt. Minden tevékenységhez szükségünk lesz egy útvonalra, így számba véve ezek a következők:

Route::get('/notes', );

Route::get('/notes/{note}', );

Route::get('/notes/create', );

Route::post('/notes', );

//edit

//update

//destroy

viszont mindezt megadhatjuk egyetlen sorral: Route::resource('/notes', );

<https://laravel.com/docs/9.x/controllers#resource-controllers> Itt olvashatjuk, hogy ez az egyetlen útvonal-deklaráció több útvonalat hoz létre az erőforráson végzett különféle műveletek kezelésére. A generált vezérlő már rendelkezik az összes ilyen művelethez mellékelt metódusokkal. Viszont két paramétert vár, a másik a resource controller, amelyet létre kell hoznunk. Hasonlóan hozzuk létre, mint a többi kontrollert, írjuk be a terminálra:

php artisan make:controller NoteController --resource

Ellenőrizzük létrejött e a fájl: app\Http\Controllers\NoteController.php

Ha végiggörgetünk a fájlon, láható, hogy minden szükséges metódus létrejött. (edit, show, update, destroy) Most térjünk vissza a web.php fájlra és mostmár meg tudjuk adni a második paramétert is:

Route::resource('/notes', NoteController::class)->middleware(['auth']);

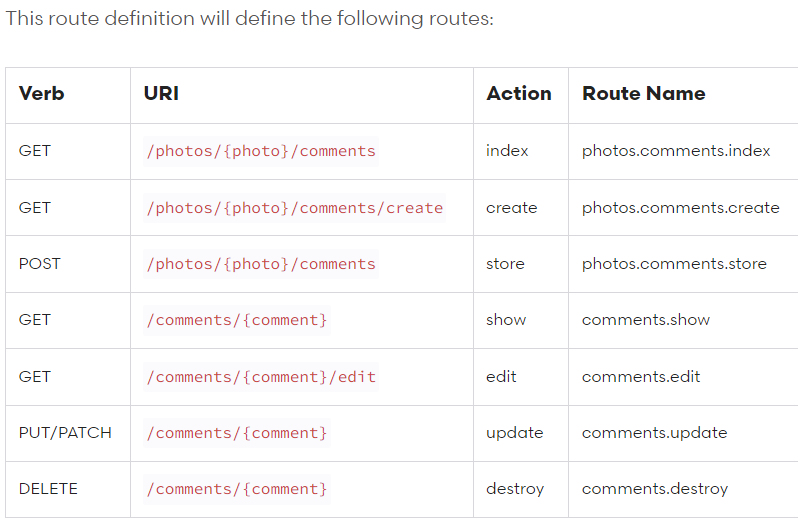
ellenőrizni kell a fájl elején hozzáadásra került e: use App\Http\Controllers\NoteController;

app\Providers\RouteServiceProvider.php fájlban a dashboardot írjuk át notes-ra

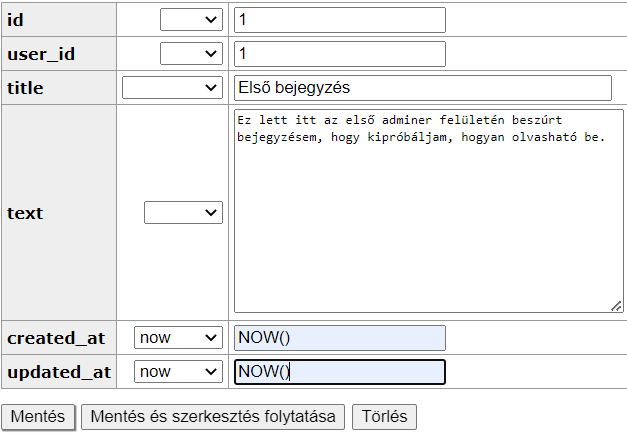
public const HOME = '/notes';

# Bejegyzések beolvasása

A jegyzetoldalunk most üres. <https://laravel.com/docs/9.x/controllers#resource-controllers>

 itt a táblázatban látható, hogy a photoes az index actionnal a kontrollerben, vagyis nyissuk meg a NoteController.php-t. az index az index metódussal lekérjük a jegyzeteket az adatbázisból, és átadjuk a tartalmat a nézetnek, hogy megjelenítsük őket.

De mivel még nincsenek jegyzeteink, adjunk hozzá manuálisan egy megjegyzést az adatbázishoz **az adminer** segítségével.

 Nyisd meg a jegyzettáblázatot, hozz létre egy új sort egy sor hozzáadásával, az id:1, a user\_id szintén 1, mert eddig csak egy felhasználónk van egy azonosítóval, címmel, adj neki egy címet pl első bejegyzés, adj hozzá szöveget, az első megjegyzés szövegét, majd adj hozzá időbélyeget a Created\_at és az updated\_at számára a NOW() függvény használatával.

OK, kész az első jegyzetünk. Úgy fogjuk a kiolvasást beállítani, hogy a felhasználók csak a saját jegyzeteiket lássák. app\Http\Controllers\NoteController.php -> index metódus

Tehát először szerezzük be a bejelentkezett felhasználó azonosítóját, ahol :

$userId = Auth::id();

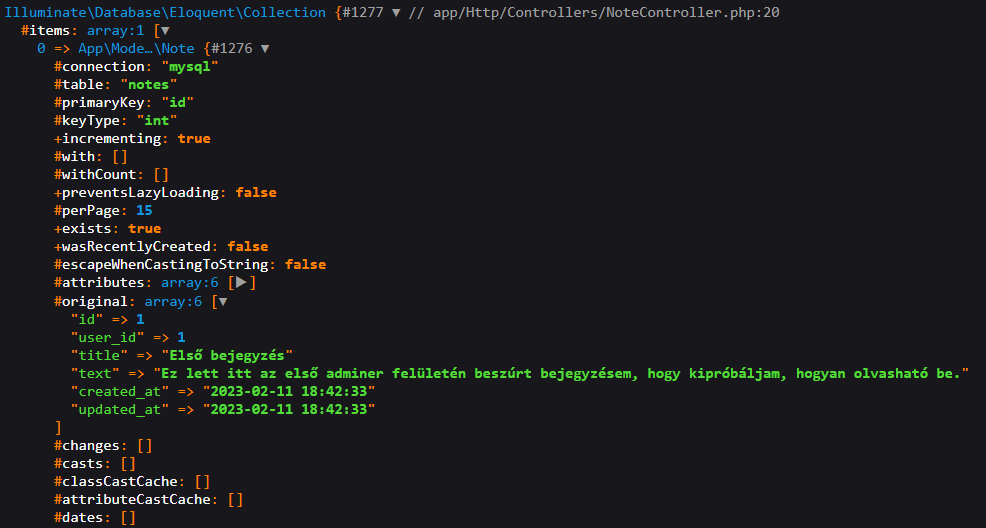
Az Auth hozzáférést biztosít a bejelentkezett felhasználóhoz. Ha jobb egérgombbal kettőt kattintunk az auth-ra mivel pirossal aláhúzza a program, akkor automatikusan hozzá adja az oldalhoz az osztály importot: use Illuminate\Support\Facades\Auth;

Most az eloquent használatával kapjuk meg ennek a felhasználónak az összes jegyzetét:

$notes = Note::where('user\_id', $userId)->get();

dd($notes);

itt is kattintsunk jobb egérgombbal a Note-ra, és osztály hozzáadásra. Nézzük meg a böngészőben:



Készítsünk még egy jegyzetet, hasonlóan, mint az előbb. Tehát adj hozzá egy második sort, a user\_id: 1 itt is. Ha újra megnézzük a böngészőt, akkor már két elem lesz a tömbben.

Írjuk át a kódunkat a következőképpen:

public function index()

{

$userId = Auth::id();

$notes = Note::where('user\_id', $userId)->get();

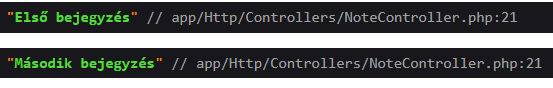
$notes->each(function($note){

dump($note->title);

});

}

Tehát a $userId-t közvetlenül az Auth::id-re cseréltük. Az egyes metódusok gyűjteményeit használva csak a jegyzetek címét jelenítsük meg.



Időrendi sorrendben jelennek meg a bejegyzések, de meg is fordíthatjuk őket:

public function index()

{

$notes = Note::where('user\_id', **Auth::id())->latest()**->get();

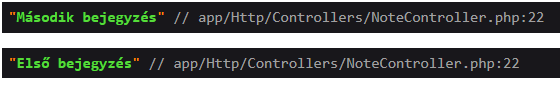
$notes->each(function($note){

dump($note->title);

});

}

Ha ráfrissítünk, látható, hogy fordított sorrendben fognak megjelenni.



Módosítsuk az adatbázisban az első bejegyzésünket a dátummal együtt.

módosíthatjuk a kódunkat úgy is, hogy mindig a legfrissebb kerüljön előre, akkor is ha az csak módosításra került.

$notes = Note::where('user\_id', Auth::id())->latest('updated\_at')->get();

Így biztosan mindig a legutolsó jegyzeteket fogja nekünk megjeleníteni. A beolvasást a tömbbel kommentezzük ki alatta, ez az egy sor maradjon ott.



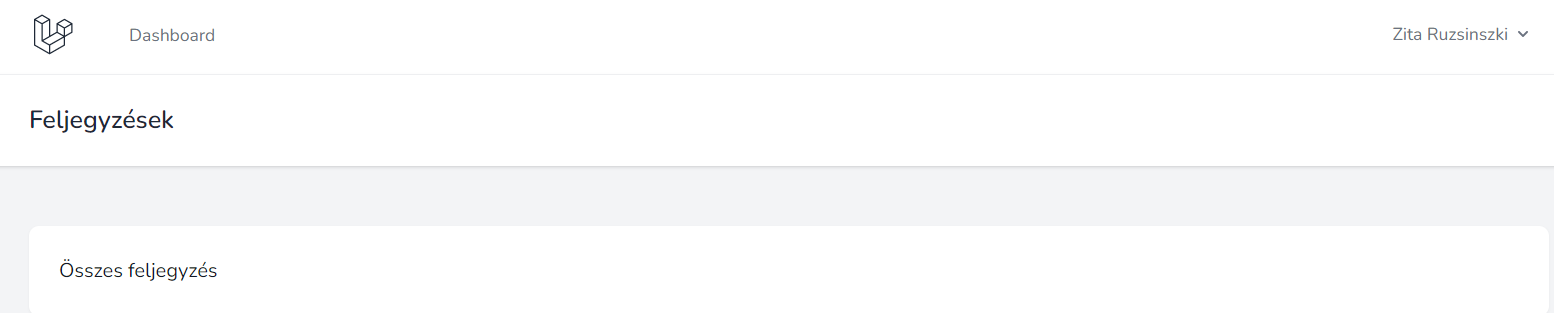
# Jegyzetek renderelése a böngészőben

Hozzunk létre a resources\views mappában egy új mappát notes néven, és ezen belül egy új fájlt: **index.blade.php**. Mivel ez az oldal is nagyon hasonló lesz a dashboard.blade.php oldalhoz, másoljunk át abból mindent ebbe. Két sort javítsunk, a 4ik sorban: {{ \_\_('Feljegyzések') }}, és 12-ik sorban: {{ \_\_("Összes feljegyzés") }}

Nyissuk meg a NoteController.php-t újra: és adjuk meg visszatérési értéknek, az újonnan létrehozott új fájlunkat:

return view('notes.index');

A böngészőben már a megjelenik az oldalunk:



Egészítsük ki a NoteController.php-t a with() helperrel:

return view('notes.index'**)->with('notes', $notes);**

az index.blade.php-t is egészítsük ki: használjuk a blade foreach direktívát, némi stílusformázással:

{{ \_\_("Összes feljegyzés") }}

@foreach ($notes as $note )

<div class="my-6 p-6 bg-white border-b border-gray-200 shadow-sm sm:rounded-lg">

<h2 class="font-semibold text-xl">{{ $note->title }}</h2>

<p class="mt-2">{{ $note->text }}</p>

</div>

@endforeach

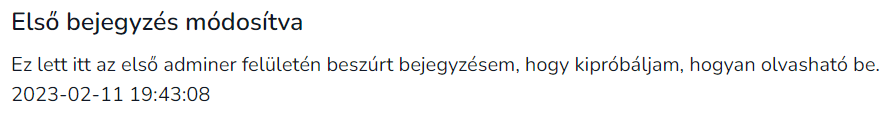
Így már a böngészőben szépen kirajzolódnak a bejegyzések:

Formázásról bővebben: <https://tailwindcss.com/>

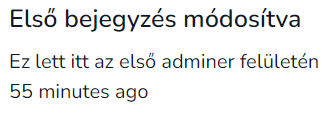
Ha kiegészítjük azzal, hogy kiírja a dátumot:

<span>{{ $note->updated\_at }}</span>

akkor a bejegyzések alatt megjelenik a létrehozás dátuma.



ezt azért felhasználó barát módon szokták inkább megjeleníteni, ezért egészítsük ki:

<span>{{ $note->updated\_at->diffForHumans() }}</span>

a diffForHumans() helper olvashatóbbá teszi a létrehozás időpontját.

Ha hozzá adjuk az @empty direktívát, megvizsgálja a tömböt, ha az üres, akkor kiírja a megadott üzenetet, de ehhez a @foreach-et le kell cserélnünk @forelse-re

**@forelse** ($notes as $note )

<div class="my-6 p-6 bg-white border-b border-gray-200 shadow-sm sm:rounded-lg">

<h2 class="font-semibold text-xl">{{ $note->title }}</h2>

<p class="mt-2">{{ $note->text }}</p>

<span clas="block mt-4 text-sm opacity-25">{{ $note->updated\_at->diffForHumans() }}</span>

</div>

**@empty**

**<p>Még nincsenek feljegyzései...</p>**

**@endforelse**

Egyenlőre, itt még nem látjuk a változást, mert vannak bejegyzések.

# Lapozó hozzáadása

A laravellben meglehetősen egyszerű a lapozó létrehozása. Nyissuk meg a NoteController.phpt, és írjuk át a get-et paginate-re, mivel nincs sok oldalnyi anyagunk, adjuk meg paraméternek, hogy 1 darab bejegyzést jelenítsen meg.

$notes = Note::where('user\_id', Auth::id())->latest('updated\_at')->**paginate(1);**

Ha ráfrissítünk a böngészőben, akkor el is tűnik a második bejegyzés.

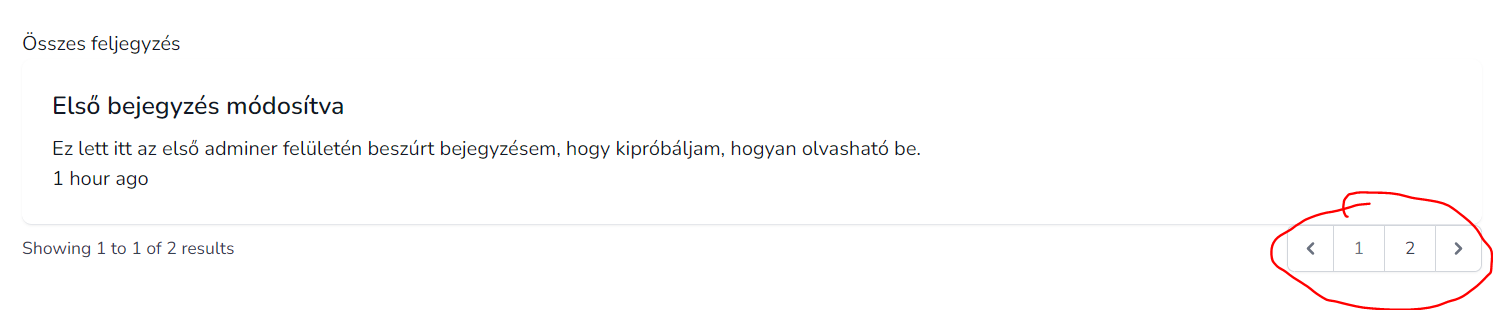
az index.blade.php-t is egészítsük ki

@endforelse

**{{ $notes->links() }}**

</div>

Ha most ráfrissítünk meg is jelenik a lapozó. Emlékeztek mennyit vacakoltunk vele az elején?



Adjunk meg az 1 helyett limitnek most 5-öt, meg is fog jelenni mindkét bejegyzés. A következő probléma, amit meg kell oldani, mi történjen, ha egy bejegyzés túl hosszú. Az adatbázisban adjunk hozzá az első bejegyzéshez, egy jó adag lorem ipsum szöveget. Limitáljuk le a megjelenítendő szöveg méretét a Limit helper segítségével, amely két paramétert fog várni tőlünk, első mit, második mennyire limitálja. Az index.blade.php oldalon kell beállítanunk:

<p class="mt-2">{{ Str::Limit($note->text, 200) }}</p>

Ha megnézzük a böngészőben láthatjuk, hogy el is tűnt a szöveg egy része.

A weboldal tetején még mindig a Dashbar szöveg szerepel, cseréljük le egy linkre.

resources\views\layouts\navigation.blade.php keressük meg itt a Dashboard szöveget és írjuk át Feljegyzésekre, a dashboard route - útvonal nevet pedig notes.index-re. Ezeket a lépéseket végezzük el a welcome.blade.php fájlban is. Használjuk itt is az URL helyett a route helper metódust: <a href="{{ **route('notes.index')** }}"

Ha most a böngészőbe csak a litenotes.test oldalt töltjük be, akkor meg is jelenik a jobb felső sarokban a Feljegyzések felirat, és a dashboard.blade.php fájlt óvatosan ki is törölhetjük.

# Új feljegyzés hozzáadása a weboldalról

Először is egészítsük ki resources\views\notes\index.blade.php fájlunkat egy új gombbal, ahol az útvonal megadásához a route helper metódust használjuk:

<a href="{{ route('notes.create') }}" class="btn-link btn-lg mb-2">+ Új bejegyzés</a>

***Ha nem működik a formázás futtasd a : npm run dev – parancsot vagy a php artisan serve***

A css-t is kiegészíthetjük:

@layer components{

.btn-link{

@apply inline-block px-4 py-2 bg-indigo-700 rounded-md font-bold text-sm text-white uppercase;

}

.btn-lg{

@apply px-6 py-3;

}

}

Megjelenik az új kék gombunk, és ha rákattintunk az url-nek is működnie kell: <http://litenotes.test/notes/create> egyenlőre egy üres oldalt kapunk. Térjünk vissza a NoteController.php odalra, ahol a create metódust egészítjük ki:

public function create()

{

return view('notes.create');

}

Ha hozzá adtuk a nézetet, készítsük el hozzá az oldalt is, a **resources\views\notes** mappán belül hozzunk létre egy új fájlt a **create.blade.php** –t. Ez is nagyon hasonló lesz mint az index fájlunk, így átmásoltam és csak a felesleget töröltem:

<x-app-layout>

<x-slot name="header">

<h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

{{ \_\_('Feljegyzések') }}

</h2>

</x-slot>

<div class="py-12">

<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

<div class="my-6 p-6 bg-white border-b border-gray-200 shadow-sm sm:rounded-lg">

</div>

</div>

</div>

</div>

</x-app-layout>

Így meg is kapjuk a kívánt formátumot. Ezen belül hozzuk létre a formot, ahol felvihetjük az új feljegyzéseket.

<form action="" method="POST">

<input type="text" name="title" placeholder="Cím">

<textarea name="text"rows="10" placeholder="Kezdj el írni ide..."></textarea></form>

Az resources\views\components\text-input.blade.php fájlban hozzáadásra került egy stílus, amit mi újrahasznosíthatunk, ha x-és fájl nevével kiegészítjük az input mezőt, de akkor zárótaget is kell hozzáadnunk így:

**<x-text-**input type="text" name="title" placeholder="Cím" **class="w-full**"> **</x-text-input>**

Így meg is kapja a mező a formázást, a w-full osztály hozzáadására kitölti a teljes képernyő szélességet. A textarea-t is igazítsuk hozzá, ehhez a component mappában létre kell hoznunk egy új fáljt: **textarea.blade.php**-t és másoljunk is át bele mindent a text-input.blade.php-ből.

De az inputot át kell írni textarea-ra és záró taget is hozzá kell adni, különben bele kerül nélküle a következő összes elem. Így:

@props(['disabled' => false])

<textarea {{ $disabled ? 'disabled' : '' }} {!! $attributes->merge(['class' => 'border-gray-300 dark:border-gray-700 dark:bg-gray-900 dark:text-gray-300 focus:border-indigo-500 dark:focus:border-indigo-600 focus:ring-indigo-500 dark:focus:ring-indigo-600 rounded-md shadow-sm']) !!}></textarea>

Tehát így:

<x-textarea name="text" rows="10" placeholder="Kezdj el írni ide..." class="w-full mt-6">

</x-textarea>

Adjunk hozzá mentés gombot, nézzük meg a componentben milyen nevű gombok vannak, és azokból választhatunk:

<x-primary-button class="mt-6" >Mentés</x-primary-button>

Az mt-6 osztály margin-top 6 px-nek felel meg.

Az felvitt adatokat el is kell tárolnunk, így az action-t egészítsük ki:

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

Próbáljunk meg a böngészőben ezek után új bejegyzést felvinni és menteni.

Nem lesz jó, mert a keretrendszer próbálja a támadásokat megelőzni. <https://laravel.com/docs/9.x/csrf> Alapvetően minden olyan űrlap, ami post, put, patch, delete metódust használ, ki van téve támadásoknak. Ezért a Laravel minden munkamenethez automatikusan generál egy CSRF tokent, hogy megbizonyosodjon arról, hogy a hitelesített felhasználó nyújtja be a kérést , nem pedig valaki kívülről. Így a Laravel minden űrlapjának tartalmaznia kell egy **CSRF** mezőt, egy rejtett mezőt a kérelem érvényesítéséhez.

<input type="hidden" name="\_token" value="{{ csrf\_token() }}" />

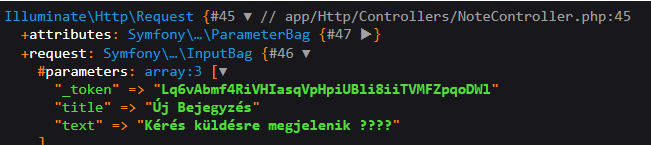
De nem kell minden alkalommal hozzáadni az egész elemet, egyszerűen használhatjuk a CSRF blade direktívát az űrlapon belül, így**:@csrf**. Adjuk hozzá az űrlaphoz az input mező elé, és máris megszűnik a hiba üzenet, ha megpróbálunk új bejegyzést hozzáadni és menteni próbáljuk.

# Űrlapon felvitt adatok mentése az adatbázisba

Térjünk vissza a NoteController.php fájlunkhoz, a store metódushoz. Először csak adjuk hozzá:

dd($request);

metódust, és ellenőrizzük megérkezik e ide a kérés. Próbálunk egy új felvitelt menteni, a böngészőben pedig meg kell jelennie:



Mielőtt elmentenénk ezeket az adatokat az adatbázisunkba, ellenőriznünk kell azokat. Ez megtehető a kérés objektum által biztosított érvényesítési módszerrel, így.

$request->validate([

'title' => 'required|max:120',

'text' => 'required'

]);

Érvényesítés kérése -> Adjuk meg az érvényesíteni kívánt mezőket kulcsként: Cím, majd a szöveg, és mint érték, az érvényesítési szabályokat. Azt akarjuk, hogy a cím mező kötelező legyen és, hogy a maximális hossz 120 karakter. Ezután azt szeretnénk, hogy a szövegmező is kötelező legyen.

Az összes elérhető érvényesítési szabály megtalálható a laravel dokumentációban itt: <https://laravel.com/docs/9.x/validation#rule-integer>

Most, ha ez az ellenőrzés sikeres, a kódunk a szokásos módon fut tovább. Ha az érvényesítés sikertelen, a felhasználót a rendszer a hibaüzeneteket tartalmazó űrlapra irányítja át , és ezek a hibaüzenetek a errors változóban érhetők el. Nyissuk meg a

resources\views\notes\create.blade.php fájlunkat, és adjuk hozzá, ha hibát talál a validálás során a program a form mezők fölé írja ki.

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

<p>{{ $error }}</p>

@endforeach

Nyissuk meg a böngészőt és kitöltés nélkül kattintsunk a mentés gombra. Szépen kiírja nekünk, hogy a két mező kitöltése kötelező. Tegyük egyértelműbbé, hogy mit hol kell kitöltenie a felhasználónak:

<x-text-input type="text" name="title" placeholder="Cím" class="w-full" aria-autocomplete="off"></x-text-input>

**@error('title')**

**<div class="text-red-600 text-sm"> {{ $message }} </div>**

**@enderror**

<x-textarea name="text" rows="10" placeholder="Kezdj el írni ide..." class="w-full mt-6">

</x-textarea>

**@error('text')**

**<div class="text-red-600 text-sm"> {{ $message }} </div>**

**@enderror**

A két űrlapmező alá helyezzük el az error blade direktívát, egy kis formázással adjuk meg most a hibaüzeneteket. Próbáljuk ki a böngészőbe, hogyan is fog ez kinézni. A mezők alatt fog pirossal megjelenni a hibaüzenet.

Viszont az lenne az ideális, ha ezeket a hibaüzeneteket áthelyezhetnénk az components-be.

Keressük meg a componentsben azokat a fájlokat, amelyekre az x-slot-tal hivatkoztunk és másoljuk át oda a hibaüzeneteket. Vagyis a title-t a text-input.blade.php-be másoljuk át, a text error-t pedig az általunk létrehozott textarea.blade.php fájlba.

Próbáluk ki, ugyanúgy meg kell kapnunk a várt eredményt. Viszont az error direktíva paramétereként jobb ha a $field változót adjuk meg mindkét esetben

@error($field)

és a @props-ot is kiegészítjük:

@props(['disabled' => false, 'field' => ''])

ugyanakkor mindkét input mezőt a create.blade.php-n ki kell egészíteni a field atribútummal, az input-ot: <x-text-input type="text" name="title" **field="title"** placeholder="Cím" class="w-full" aria-autocomplete="off"></x-text-input> a textareát pedig a **field="text"-**el.

Böngészőben ha ellenőrizzük, vagyis elpostoljuk az üres űrlapot ugyanúgy meg kell kapnunk az hibaüzeneteket.

Következő probléma, ha kipróbáljuk, hogy a címet beírjuk, de a textareát nem töltjük ki. Akkor csak a texareához adja hozzá a hibát, de eltűnik a címből a szöveg. Jó lenne a már helyes szöveget megőrizni, ehhez az ún. old blade direktívát kell használnunk, hogy be tudjuk állítani a meglévő mezők értékét. Az input mezőhöz adjuk hozzá:

:value=”@old(’title’)”

A textareához:

:value="@old('text')"

a textarea.blade.php-ben is meg kell adni, hogy olvassa be a beírt értéket:

@props(['disabled' => false, 'field' => '', **'value' => ''**])

…**{{ $value }}** </textarea>

és próbáljuk ki, hogy a címet üresen hagyjuk, és csak a textbe írunk. Ha jól működik, töltsük ki mindkettőt és mentsük. A notes oldalra jutunk, mivel mindkét mező valid volt.

Térjünk vissza a NoteController.php store metódusához: használjuk az eloquent buildert és a save metódust:

$note = new Note([

'user\_id' => Auth::id(),

'title' => $request->title,

'text' => $request->text

]);

$note->save();

de ezt megadhatjuk elegánsabban is, hogy használjuk a Note modellt:

Note::create([

'user\_id' => Auth::id(),

'title' => $request->title,

'text' => $request->text

]);

Ez a formátum automatikusan létrehozza és elmenti az új objektumot. Mentés után szeretnénk visszatérni az indexhez:

return to\_route('notes.index');

A to\_route metódust csak nemrég adták hozzá a Laravel9 verziójához.

Ha kipróbáljuk egy újabb hiba üzenetet kapunk: MassAssignmentException-t. Ez ismét a Laravel egy másik biztonsági funkciója.

Keressük meg a Mass Assignment elemet a dokumentumokban.

<https://laravel.com/docs/9.x/eloquent#mass-assignment>

Az Eloquent modellek alapértelmezés szerint védettek a tömeges hozzárendelési sebezhetőségekkel szemben. Ennek megkerüléséhez nyissuk meg: **app\Models\Note.php** fájlunkat.

Itt két lehetőség közül választhatunk. Használhatjuk a $fillable védett változót, és megadhatjuk az összes szükséges mezőt, vagy használunk helyette egy védett változót:

{ use HasFactory;

protected $guarded = []; }

Frissítsünk a böngészőre és megjelenik az előzőleg elpostolt üzenetünk, próbáljunk meg hozzá új bejegyzést adni a web felületről. És nagyszerű, elmentettük első jegyzetünket az alkalmazáson keresztül.

# Bejegyzések címének linkké alakítása

Első lépésként az index.blade.php oldalon írjuk át a címet:

**<a href="{{ route('notes.show', $note->id) }}">** {{ $note->title }} **</a>**

frissítsünk a böngészőben és kattintsunk valamelyik bejegyzés címére. Igaz üres oldalra kerülünk, de a linkben meg fog jelenni egy egyedi azonosító, id.

A megjelenéshez a NoteController.php show metódusát kell kitöltenünk, amely már kapott paramétert:

public function show($id)

{

$note = Note::where('id', $id)->where('user\_id', Auth::id())->firstOrFail();

return view('notes.show')->with('note', $note);

}

Hozzuk létre ennek megfelelően : resources\views\notes\**show.blade.php** fájlt, amelynek a tartalma nagyjából ugyanaz lesz, mint az index.blade.php-nek, másoljuk át és vegyük ki a limitet, hogy a teljes szövegek megjelenjenek.

<x-app-layout>

<x-slot name="header">

<h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">

{{ \_\_('Feljegyzések') }}

</h2>

</x-slot>

<div class="py-12">

<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

**<div class='flex m-5'>**

**<p class="opacity-70">**

**<strong>Létrehozva:</strong> {{ $note->created\_at->diffForHumans() }}**

**</p>**

**<p class="opacity-70 ml-8">**

**<strong> Frissítve:</strong> {{ $note->updated\_at->diffForHumans() }}**

**</p>**

**</div>**

<div class="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">

**<h2 class="font-semibold text-xl p-6">**

**{{ $note->title }}**

**</h2>**

**<p class="mt-2 p-6 whitespace-pre-wrap">{{ $note->text}}</p>**

</div>

</div>

</div>

</x-app-layout>

Ha mégnézzük a böngészőben az egyedi azonosítókhoz szépen megjelennek a bejegyzések, létrehozás dátuma, frissítés és a sortöréseket és új bekezdéseket is ugyanúgy tördeli, mit ahogyan az eredetileg beillesztettük. Az url-ben ha olyan számot adunk meg, amilyen id létezik betölti a hozzá tartozó bejegyzést, viszont ha olyan számot adunk meg, amelyik nem létezik, 404-es hibát látjuk, oldal nem létezik.

# Egyedi azonosítók

A kontroller show metódusában elfogadtuk az adott modell azonosítóját vagy UUID-jét , majd lekérdeztük az adatbázist, hogy megkapjuk az eredményeket. Ez a lépés teljesen elkerülhető, ha magát a jegyzetet közvetlenül küldjük be. Ekkor nem kell lekérdeznie a jegyzetet.

Távolítsuk el, az összes fejegyzést az adatbázisból. (http://litenotes.test/adminer.php)

A terminálon írjuk be:

php artisan migrate:rollback

Frissítsünk az oldalra, lehet ott is kattintani kell a migrációra.

Keressük ki a database\migrations mappán belül a létrehozott migrációs táblánkat és egészítsük ki az id() után:

$table->id();

**$table->uuid('uuid');**

Mentsük és a terminálba írjuk be:

**php artisan migrate**

ha erre azt mondaná nincs mit migrálni akkor:

**php artisan migrate:refresh**

Nézzük meg létrehozta e a migráció az adatbázis táblában az uuid oszlopot. Ha megvan akkor a következő lépés, hogy a NoteController.php-t is egészítsük ki, hogy amikor létrehozunk egy fájlt, akkor importálja be az uuid-t.

Note::create([

**'uuid' => Str::uuid(),**

Jelezni fogja a Str-re, hogy hiányzik az osztály, jobb egérgombbal hozzá tudjuk adni, ki kell választani illiminate…stb. A böngészőben még mindig az id jelenik meg, ezért nyissuk meg az index.blade.php-t és javítsuk az id-t uuid-ra: <a href="{{ route('notes.show', **$note->uuid**) }}">{{ $note->title }}</a> Ezért a NoteController.php fájl show metódusát is javítani kell:

public function show**($uuid)**

{

$note = Note::where(**'uuid', $uuid)->**where('user\_id', Auth::id())->firstOrFail();

return view('notes.show')->with('note', $note);

}

Ha újra viszünk fel bejegyzést, és megnézzük a böngészőben (lehet újra kell regisztrálni) akkor a böngésző url-jében egy hosszú egyedi azonosítót fogunk látni. Itt lekérdezésre kerül az id alapján, de ezt a lépést kihagyhatjuk, ha átírjuk így:

public function show(Note $note)

{

return view('notes.show')->with('note', $note);

}

Nem kell külön lekérdeznünk a bejegyzést, de természetesen ezt az index.blade.php fájlban is javítani kell:

<a href="{{ route('notes.show', **$note**) }}">{{ $note->title }}</a>

Ha visszatérünk a böngészőben a bejegyzésekhez, és ráfrissítünk, és újra rákattintunk, akkor megint az id-je jelenik meg a böngészőben. Ezt úgy hívjuk route modell név alapján, útvonal kötése a névhez. Magát a teljes modellt beinjektáljuk vagy kötjük az útvonalunkhoz. Természetesen itt látjuk az elsődleges kulcsunkat, mert ez az alapértelmezett kulcs a modell feloldásához. A testreszabásához nyissuk meg a **app\Models\Note.php** fájlunkat és adjunk hozzá egy metódust:

protected $guarded = [];

**public function getRouteKeyName(){**

**return 'uuid';**

**}**

a metódus hozzáadásához megadtuk a nyilvános getRouteKey, getRouteKeyName függvényt, és ez vissza adja az UUID-t, a modell feloldásához használni kívánt kulcsot. Most frissítsük a böngészőt, és újra visszakaptuk az UUID-t a linkben.

Van azonban egy problémánk. Természetesen itt elkerültük az adatbázis lekérdezését, de most egy felhasználó megtekintheti egy másik felhasználó megjegyzését, ha megkapja az UUID-t. Ezt meg kellene akadályozni. A NoteController-php- oldalon a show metódust egészítsük ki:

**if($note->user\_id != Auth::id()){**

**return abort(403);**

**}**

return view('notes.show')->with('note', $note);

Ha a jegyzet user\_id értéke nem egyenlő az Auth::id értékkel, akkor adja vissza a 403-as oldalt hozzáférés megtagadva hibával. Ellenkező esetben folytatjuk, és visszatérünk a nézetünkhöz. Ellenőrizzük, hogy ez működik-e. Nyisd meg a böngészőt, másold ki ezt az UUID-t, jelentkezz ki, és regisztrálj egy új felhasználót. Próbáld meg megnézni a jegyzetet, és közvetlenül illeszd be az UUID-t az urlbe. Ha minden jól működik egy 403-ast kapunk, mert nem tudunk hozzáférni egy másik felhasználó bejegyzéséhez. Ezt egy egyszerű elágazással oldottuk meg, de erre a Laravel rendelkezik szabályokkal, meghatározhatunk házirendeket, és megadhatjuk, hogy a felhasználó jogosult-e bizonyos műveletek végrehajtására vagy sem, haladó szinten.

# Bejegyzések szerkesztése

Mostmár a userünk meg tudja nézni a bejegyzéseit, adjunk az egyedi nézethez egy szerkesztés gombot. Nyissuk meg a **resources\views\notes\show.blade.php** fájlt. Adjuk hozzá az időbélyegek után:

<a href="{{ route('notes.edit', $note) }}" class="btn-link ml-auto">Szerkesztés</a>

a használt route modell miatt nincs szükségünk az id megadására, hogy az adott bejegyzés kerüljön szerkesztésre. Ha a stílus nem tölt be futtassuk az npm run dev parancsot a terminálban.

A szerkesztés gombra kattintva megkapjuk a uuid-t\edit-et az urlben.

Nyissuk meg újra a NoteController.php-t és folytassuk az edit metódus kiegészítésével:

public function edit(Note $note)

{

if($note->user\_id != Auth::id()){

return abort(403);

}

return view('notes.edit')->with('note', $note);

}

Majd készítsük kel hozzá a **resources\views\notes** mappában az **edit.blade.php** fájlt.

Másoljuk át bele a create.blade.php fájl tartalmát.

Módosítsuk a szükséges mezőket:

Headerben:

{{ \_\_('Szerkesztés') }}

Form:

Most a metódus értékénél, ha megnézzük a laravel frissítési művelet erőforrásvezérlő-dokumentációját, látjuk, hogy az vagy put, vagy patch. De a HTML űrlapok nem tudnak put javítást vagy törlési kérelmet végrehajtani. Tehát megtartjuk a postot úgy, ahogy van, de megadjuk a blade metódus direktívát, a kimeneti érték használatával. Tehát ez nagyon hasonlóan működik a CSRF blade direktívához egy rejtett beviteli mező hozzáadásával. Ekkor a bemeneti értéknek valójában a szerkeszteni kívánt cím meglévő értékének kell lennie.

<form action="{{ route('notes.update', $note) }}" method="post">

@method('put')

Value:

:value="@old('title', $note->title)"></x-text-input>

és

:value="@old('text', $note->text)"></x-textarea>

A NoteController.php update metódusát egészítsük ki:

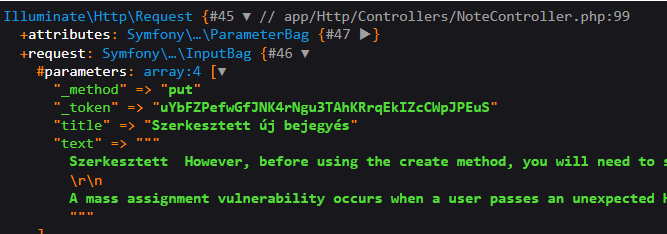
public function update(Request $request, $id)

{

dd($request);

}

Próbálunk ki egy szerkesztést, és nézzük meg megkapjuk e a böngészőben a szerkesztett adatokat.

 kommentezzük ki a dd()-t.

Először is szükségünk lesz az authorizációra, hogy a felhasználónk belépett és jogosan használja az oldalt, másoljuk át, majd a store-ból a validálást, és az update:

public function update(Request $request, Note $note)

{

// dd($request);

if($note->user\_id != Auth::id()){

return abort(403);

}

$request->validate([

'title' => 'required|max:120',

'text' => 'required'

]);

$note->update([

'title' => $request->title,

'text' => $request->text

]);

return to\_route('notes.show', $note);

}

Frissítsük a böngészőt és próbáljuk ki.

# Törlés

A szerkesztést a **resources\views\notes\show.blade.php** fájlban kezdjük. A szerkesztés gomb mellé teszünk egy törlés gombot, de ez másképp fog működni mint a szerkesztés.

<form action="{{ route('notes.destroy', $note) }}" method="post">

@method('delete')

@csrf

<button type="submit" class="btn btn-danger ml-4">Törlés</button>

</form>

A css-hez is tudunk hozzáadni némi stílust:

.btn{

@apply px-4 py-2 rounded-md font-bold text-sm text-white uppercase tracking-widest focus:ring-2

}

.btn-danger{

@apply bg-red-600 hover:bg-red-700 active:bg-red-700 ring-red-300

}

A törlés gombra kattintva, szeretnénk, ha a user megerősítené, hogy valóban szeretné törölni az adott bejegyzést. Egészítsük ki a submit gombunkat:

<button type="submit" class="btn btn-danger ml-4" **onclick="return confirm('Biztosan törölni szeretné a bejegyzését?')**">Törlés</button>

Hajtsuk végre valóban a törlési műveletet, térjünk vissza a NoteController.php oldalra, és folytassuk a destroy metódus kiegészítésével.

public function destroy(Note $note)

{

if($note->user\_id != Auth::id()){

return abort(403);

}

$note->delete();

return to\_route('notes.index');

}

Frissítsük a böngészőt, és az újabb törlési kérés után valóban el tűnik a bejegyzés, amit töröltünk.

# Visszajelzés a usernek a végrehajtott művelet sikerességéről

Első lépésként egészítsük ki a NoteController.php update metódusát:

return to\_route('notes.show', $note**)->with('success', 'A bejegyzés frissítése megtörtént');**

Nyissuk meg újra a show.blade.php fájlt és egészítsük ki, ha sikeres a munkamenet, akkor írja ki a siker esetére megadott üzentet:

<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

@if(session('success'))

<div class="mb-4 px-4 py-2 bg-green-100 border border-green-200 text-green-400 rounded-md">

{{ session('success') }}

</div> @endif

Próbáljuk ki, hogy módosítunk egy bejegyzést. Egy szép zöld keretes üzentet kell, hogy kapjunk a sikeres műveletről.

Hozzunk létre a component mappában egy új fájlt**: alert-succes.blade.php** és az előző helyről vágjuk ki és másoljuk ide bele a kódunkat egy sort kicserélve benne:

@if(session('success'))

<div class="mb-4 px-4 py-2 bg-green-100 border border-green-200 text-green-400 rounded-md">

**{{ $slot }}**

</div>

@endif

a helyére pedig írjuk be:

<x-alert-success>

{{ session('success') }}

</x-alert-success>

A másik esemény, amiről megerősítést adhatunk a végrehajtás sikeréről a felhasználónak az a törlés. Térjünk vissza NoteController.php oldalra, a destroy metódushoz és egészítsük ki egy üzenettel:

return to\_route('notes.index'**)->with('success', 'A bejegyzés törölése sikeres');**

Majd az index.blade.php fájlt egészítsük ki:

<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

<x-alert-success>

{{ session('success') }}

</x-alert-success>

Ha kipróbáljuk, akkor itt is megjelenik a törlés utána az üzenet a törlés sikerességéről.

# Eloquent kapcsolatok

Az adatbázistáblák gyakran kapcsolódnak egymáshoz.

post table = likes table

students table = subjects table

users table = notes table

Például egy bejegyzésnek sok lájkja lehet, egy diák sok tárgyat felvehet, esetünkben pedig egy felhasználónak sok jegyzete lehet. Ha túl sok táblája van, és több reláció van a táblák között, mint ez, akkor a lekérdezés, bonyolulttá válhat. Az Eloquent megkönnyíti az ilyen kapcsolatokkal való munkát, és számos típusú kapcsolatot is támogat, például egy az egyhez, egy a sokhoz, egy a sok közül, sok a sokhoz, és az egyéni típusokat is. Esetünkben a felhasználói modell egy a sokhoz viszonyban áll a jegyzetmodellel. Ezzel az alapismerettel nézzük hogyan lehet definiálni az ékesszóló kapcsolatokat , majd használni őket a modellek lekérdezéséhez és mentéséhez.

Adjon hozzá egy szerkesztési módot és nézetet a jegyzet szerkesztésének engedélyezéséhez

Ha ebben a szakaszban kiválasztja az átírási sorokat, az időbélyegzőhöz navigál a videóban

Miután a modellosztályokban kapcsolatokat definiáltunk, hozzáférhetünk egy kapcsolódó modellhez, akárcsak a tulajdonságaihoz.

Kezdjük a user modellel. Nyisd meg az alkalmazást, modell user.php fájlt. Ez a user modell a note modellhez kapcsolódik. A kapcsolat meghatározásához adjunk hozzá egy public funcitont a fájl végére.

public function notes(){

return $this->hasMany(Note::class);

}

jobb egérgombbal importáljuk az osztályt.

Mivel a felhasználói modellnek egy a többhöz kapcsolata van, ami azt jelenti, hogy egy felhasználónak sok jegyzete lehet, meg kell hívnunk a a hasMany metódust, ennél a modellnél ennek sok van, és át kell adnunk a note modellt első argumentumként és visszaadja az eredményt.

Egy az egyhez kapcsolat esetén a hívásnak hasOne, ahelyett, hogy hasMany, és ezt is módosíthatjuk egyes számra note()-ra. Ez a definíció lehetővé teszi számunkra, hogy hozzáférjünk egy adott felhasználó jegyzeteihez, mintha a jegyzetek a modellben meghatározott tulajdonságok lennének.

Mielőtt azonban ezt ténylegesen megtennénk, definiáljuk azt az inverz kapcsolatot is , amely lehetővé teszi, hogy hozzáférjünk egy adott hangjegy felhasználójához. Ezt a jegyzetmodellben lehet megtenni. Nyissuk meg a note.php-t. Hasonlóan ahhoz, amit a user modellben tettünk,

public function user(){

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

}

Ez szinguláris azért csak user() és nem users, mert itt ez a fordított kapcsolat, egy felhasználónak lehet sok bejegyzése, mivel ez az inverz reláció. Ide is importálhat osztályt. Mentse el ezt a fájlt, mentse el ezt a fájlt is. Ezzel a módszerrel elérhetjük a jegyzet felhasználóját, akárcsak a jegyzetmodell bármely más tulajdonságát.

Most definiáltuk egy a sokhoz kapcsolati és felhasználói modellünket a hasMany funkcióval.

Ha megnyitjuk a NoteController.php fájlunkat az index metóduson belül láthatjuk, hogy a bejegyzések lekérdezését úgy oldottuk meg, hogy a legutolsó legyen mindig legelöl.

Most egyszerűen lecserélhetjük mindezt az Auth::user-re, először lekérjük a felhasználót , és lekérjük a kapcsolatot használó felhasználó összes megjegyzését , majd hozzáadjuk ugyanazt a lekérdezést.

$notes = Auth::user()->notes()->latest('updated\_at')->paginate(5);

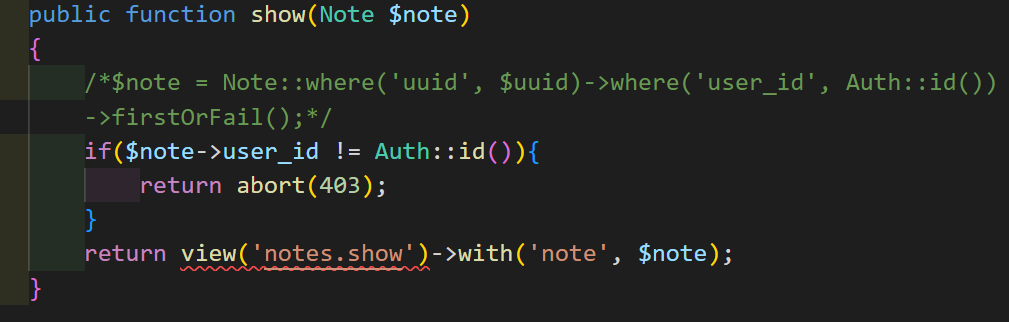
A notes()-ot alá fogja húzni a VSC, de jó lesz. Frissítsd az oldalt, és látni fogod, hogy semmi sem változott.

Másik módszer, tedd kommentbe az előzőt.

$notes = Note::whereBelongsTo(Auth::user())->latest('updated\_at')->paginate(5);

A Laravel8 verziója óta használhatjuk az inverz relációt is , és ezt a megjegyzésekkel helyettesíthetjük = Note::whereBelongsTo whereBelongsTo Auth::user , majd ugyanazt a lekérdezést használhatjuk. Ellenőrizzük ezt is, nyisd meg a böngészőt, frissíts, ismét továbbra is a felhasználó legfrissebb bejegyzései láthatóak.

Lássuk, hol tudjuk még hasznosítani a kapcsolatokat. Görgess le a show metódushoz, ahol az id-t használjuk, és ellenőrzik az idegen kulccsal.



Most az idegenkulcs ellenőrzés helyett, közvetlenül hozzáférhetünk a notes userhez , mintha az a notes modell egy tulajdonsága lenne, majd azt is() függvény használatával összehasonlíthatjuk az authorized felhasználóval, és itt átadhatjuk az auth felhasználót.

**if(!$note->user->is(Auth::user())){**

**return abort(403);**

**}**

return view('notes.show')->with('note', $note);

Ez a módja annak, hogy összehasonlítsuk az azonos modellosztályba tartozó két objektum elsődleges kulcsát. Tehát ha a jegyzet nem tartozik a felhasználóhoz és nem azonos a létrehozóval, akkor nem lesz jogosult a megtekintésre. Az összes előfordulási helyen az editnél stb. lecserélhetjük erre, de az előző megoldás is jó, hagyd azt, amelyik Neked szimpatikusabb vagy érthetőbb. Másold ki egy bejegyzés uuid-ját és próbáld más user profilból megnézni.

A NoteController.php-ben

Note::create([

'uuid' => Str::uuid(),

**'user\_id' => Auth::id(), <- itt ellenőrzésre kerül a user id,**

'title' => $request->title,

'text' => $request->text

]);

Ha átírjuk így, azt a sort kitörölhetjük:

**Auth::user()->notes()->create([**

'uuid' => Str::uuid(),

'title' => $request->title,

'text' => $request->text

]);

# Soft delete – azaz a törlés után visszaállítható legyen még a jegyzet

Ezen a ponton, amikor törlünk egy jegyzetet, az véglegesen eltűnik. De ideális esetben azt szeretnénk, ha a jegyzetek a kukába kerülnének, hogy a felhasználó később visszakereshesse vagy véglegesen eltávolíthassa őket. Az Eloquent lehetőséget kínál erre is. Először is hozzá kell adnunk egy "Törölve" oszlopot a jegyzettáblázathoz. Így, amikor törölünk egy rekordot, ez az oszlop frissül a törlés időpontjának időbélyegével. Ellenkező esetben az oszlop üres. Tehát nyissuk meg a terminált, és hajtsunk végre egy új migrációt.

php artisan make:migration add\_soft\_delete\_to\_notes

nézzük meg a Database – miration mappában létre jött e az aktuális dátummal egy új fájl, amelyben 2 functiont láthatunk egy up és egy down, amelyben a tábla név is létrehozásra került Az up metódusba adjuk meg:

$table->SoftDeletes();

ez létre hozzá a törlés oszlopot nekünk.

A down metódusba pedig:

$table->DropSoftDeletes();

majd térjünk vissza a terminálhoz és végezzük el a migrálást:

php artisan migrate

nézzük meg az adatbázisunkat, ahol létre jön egy delete\_at oszlop

A note.php-ben adjuk hozzá a fájl elején:

use Illuminate\Database\Eloquent\SoftDeletes;

class Note extends Model{

use HasFactory;

use SoftDeletes;

Ha most megpróbálunk törölni egy bejegyzést, akkor eltűnik a bejegyzés, és azt az üzenetet adja, hogy törölve lett, de ha most megnézzük az adatbázisban, akkor látszik nem tűnik el, csak bekerül a törlés dátuma az új delete\_at oszlopba.

Térjünk vissza a **resources\views\notes\show.blade.php** fájlhoz. Írjuk át a törlés gomb feliratát arra, hogy áthelyezés a kukába, vagy valami hasonlót, a 29-ik sor környékén.

A NoteControllerben is írjuk át az üzenetet:

$note->delete();

return to\_route('notes.index')->with('success', **'A feljegyzés a kukába került'**);

Ahogy az index oldalon is látható, a törölt jegyzetünk nem jelenik meg, ami azt jelenti, hogy a jegyzetvezérlőnk index metódusában a legfrissebb metódus vagy a korábban használt get metódus nem tölti le a „soft deleted” rekordokat. A kérdés az, hogyan szerezzük vissza őket? Először is hozzunk létre egy másik vezérlőt a kukába helyezett jegyzetekhez. Tehát nyissuk meg a terminált, és hajtsuk végre artisan parancsot.

php artisan make:controller TrashedNoteController

Létrejött az új kontrollerünk, készítsünk benne egy új index metódust:

public function index(){

return "Trash";

}

Készítsük el hozzá az útvonalat a web.php fájlban.

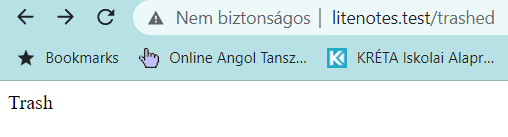
Route::get('/trashed', [TrashedNoteController::class, 'index'])->middleware('Auth')->name('trashed.index');

Hozzá rendeltük a köztes szoftvert is, mert csak a bejelentkezett felhasználóknak kell hozzáférniük ehhez az útvonalhoz, és adtunk neki nevet is, trashed.index. A félreértés elkerülése érdekében ugyanazt az elnevezési mintát követjük, mint a notes útvonalainál.

Ne felejtsük el az osztályt beimportálni, jobb egérgombbal kattintunk a nevére és import class. Ellenőrizzük a fájl elején, hogy valóban importálásra került e:

use App\Http\Controllers\TrashedNoteController;

Ellenőrizzük a böngészőben megjelenik e a beírt szöveg:

szuper az route jól működik

Most pedig próbáljuk meg vissza szerezni a kukából az adatokat. Nyissuk meg a app\Http\Controllers\TrashedNoteController.php fájlt, töröljük ki a return ’trashed’ parancsot és a NoteController index metódusából másoljuk át:

$notes = Note::whereBelongsTo(Auth::user())->**onlyTrashed()**->latest('updated\_at')->paginate(5);

return view('notes.index')->with('notes', $notes);

Csak egy aprósággal kell kiegészíteni az onlyTrashed metódussal, és megadni a return értékeit. Nézzük meg újra a böngészőben mit kapunk vissza, ahol szépen meg kell jelennie a kukába helyezett bejegyzésnek. Mostmár, csak a gomb és az oldal feliratát kell kicserélnünk, hogy a felhasználó tudja, ez a bejegyzés a kukában van. Nyissuk meg a **resources\views\notes\index.blade.php** fájlt, egészítsük ki a 4-ik sort:

{{ request()->routeIs('notes.index') ? \_\_('Feljegyzések') : \_\_('Kuka')}}

Itt tulajdonképpen ellenőrizzük, hogy az útvonal jegyzet vagy trash. Ez alapján visszaküldjük a megfelelő szöveget. Tehát itt a request metódust használjuk az útvonal ellenőrzéséhez. Az útvonal kérése a notes.index. Ha true a conditional operátor kiértékelése, akkor a Feljegyzések feliratot jeleníti meg, ha False akkor a Kukát.

Az itt található kettős aláhúzás a szöveg fordítására szolgál, ha több nyelvet használnánk a programban.

És itt lent, csak akkor jelenítsük meg ezt a gombot, ha az útvonal a notes.index. Tehát még egyszer használjuk az if blade direktívát annak ellenőrzésére, hogy a kérelem útvonala notes.index-e és végül zárjuk endif-el.

</x-alert-success>

**@if(request()->routeIs('notes.index'))**

<a href="{{ route('notes.create') }}" class="btn-link btn-lg mb-2">+ Új bejegyzés</a>

**@endif**

még egy csere:

**@if(request()->routeIs('notes.index'))**

<p>Még nincsenek feljegyzései...</p>

**@else**

**<p>Nincs feljegyzés a kukában...</p>**

**@endif**

A felirat cserélődött kukára, és eltűnt az új feljegyzés gomb is, és felül ha a feljegyzések feliratra kattintunk, akkor szépen vissza navigál a feljegyzésekhez.

Adjuk hozzá a navigációhoz a kuka feliratot is, keressük meg a **resources\views\layouts\navigation.blade.php** fájlt. Másoljuk le az ott lévő linket, saját maga alá és a notes-t javítsuk: trashed-re:

<x-nav-link :href="route('trashed.index')" :active="request()->routeIs('trashed.index')">

{{ \_\_('Kuka') }}

</x-nav-link>

Ha megnézzük a böngészőben, megjelenik az új menüpontunk.

Ugyanezt a lépést végezzük el a reszponzív nav-ban is: 71-ik sor környékén

*<!-- Responsive Navigation Menu -->*

<div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">

<div class="pt-2 pb-3 space-y-1">

<x-responsive-nav-link :href="route('notes.index')" :active="request()->routeIs('notes.index')">

{{ \_\_('Feljegyzések') }}

</x-responsive-nav-link>

**<x-responsive-nav-link :href="route('trashed.index')" :active="request()->routeIs('trashed.index')">**

**{{ \_\_('Kuka') }}**

**</x-responsive-nav-link>**

Ellenőrizzük le a működését mobil nézetben, a hamburger ikonra kattintva.

Még mindig nem tökéletes a programunk, hiszen ha a törölt feljegyzésre kattintunk akkor 404-re kiáll.

Nyissuk meg a web.php-t és adjunk hozzá egy új útvonalat, lemásolhatjuk az előző alapját, csak pár dolgot módosítunk benne:

Route::get('/trashed/{note}', [TrashedNoteController::class, 'show'])->middleware('auth')->name('trashed.show');

Ezt követően a **TrashedNoteControllerhez** adjunk hozzá egy új show metódust:

public function show(Note $note){

dd($note);

}

Az index.blade.php-ben is meg kell adni az @if-el az útvonal választását: közben nagyon figyeld meg, hogy az anchor tag hogyan kerül kihagyásba a feltételből, ha nem hagyjuk ki nem fog működni a link:

<h2 class="font-semibold text-xl">

**<a**

**@if(request()->routeIs('notes.index'))**

href="{{ route('notes.show', $note) }}"

**@else**

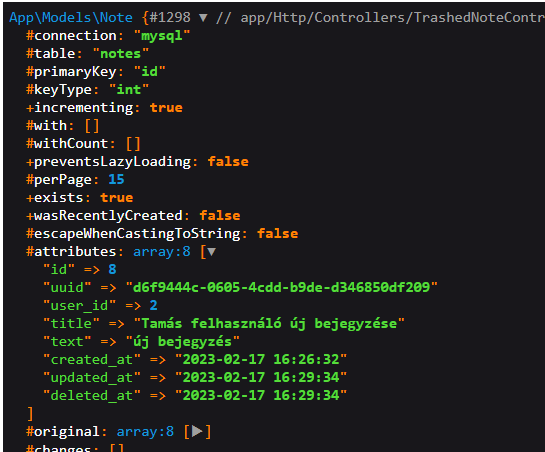
href="{{ route('trashed.show', $note) }}"

**@endif**

**>{{ $note->title }}</a>**

</h2>

Kattintsunk a kuka menüpontra, majd a kukába helyezett feljegyzés címsorára, ki fogja írnia böngészőben a tömböt, hogy mi található benne.



Térjünk vissza a TrashedNoteControllerhez, és kommentezzük ki a dd()-t, és egészítsük ki a show metódust, odafigyelve arra, hogy csak belépett felhasználó férhessen hozzá, ezt a NoteController show metódusából átmásolhatjuk:

public function show(Note $note){

//dd($note);

**if(!$note->user->is(Auth::user())){**

**return abort(403);**

**}**

**return view('notes.show')->with('note', $note);**

}

Ellenőrizzük a böngészőben, viszont a létrehozás és módosítás dátumát, a gombokat és a feliratot is kellene lecserélni.

Kezdjük a felirattal a **resources\views\notes\show.blade.php** fájlban, itt is használjunk ternary operátoros vizsgálatot:

**{{ !$note->trashed() ? \_\_('Feljegyzések') : \_\_('Kuka') }}**

Vagyis ha nincs kukában, akkor a feljegyzések felirat jelenjen meg, egyébként meg a kuka.

Az oldalt is eléggé át kell alakítani:

<div class='flex m-5'>

**@if(!$note->trashed())**

<p class="opacity-70">

<strong>Létrehozva:</strong> {{ $note->created\_at->diffForHumans() }}

</p>

<p class="opacity-70 ml-8">

<strong> Frissítve:</strong> {{ $note->updated\_at->diffForHumans() }}

</p>

<a href="{{ route('notes.edit', $note) }}" class="btn-link ml-auto">Szerkesztés</a>

<form action="{{ route('notes.destroy', $note) }}" method="post">

@method('delete')

@csrf

<button type="submit" class="btn btn-danger ml-4" onclick="return confirm('Biztosan szeretné a kukába helyezni a bejegyzését?')">Áthelyezés a kukába</button>

</form>

**@else**

**<p class="opacity-70 mr-10">**

**<strong>Törölve:</strong> {{ $note->deleted\_at->diffForHumans() }}**

**</p>**

**<form action="{{ route('trashed.update', $note) }}" method="post" class="ml-auto">**

**@method('put')**

**@csrf**

**<button type="submit" class="btn-link ml-auto">Feljegyzés visszaállítása a kukából</button>**

</form>

*<!--<form action="{{ route('notes.destroy', $note) }}" method="post">*

*@method('delete')*

*@csrf*

*<button type="submit" class="btn btn-danger ml-4" onclick="return confirm('Biztosan törölni szeretné a bejegyzését?')">Áthelyezés a kukába</button>-->*

</form>

**@endif**

TrashedNoteController-ben meg kell adni az update-et:

public function update(Note $note){

if(!$note->user->is(Auth::user())){

return abort(403);

}

$note->restore();

return to\_route('notes.show', $note)->with('success', 'Feljegyzés visszaállítva');

}

és a web.php-ben az új útvonalat:

Route::get('/trashed/{note}', [TrashedNoteController::class, 'show'])->withTrashed()->middleware('auth')->name('trashed.show');

Route::put('/trashed/{note}', [TrashedNoteController::class, 'update'])->withTrashed()->middleware('auth')->name('trashed.update');

# Végleges törlés

A feladat, hogy amikor a kuka menüpontra kattintunk és kiválasztunk egy kukába helyezett feljegyzést, jelenjen meg a visszaállítás gomb mellett egy végleges törlés gomb. Nyissuk meg újra a show.blade.php fájlt, és a kommentbe hagyott linket alakítsuk át.

<form action="{{ route('trashed.destroy', $note) }}" method="post">

@method('delete')

@csrf

<button type="submit" class="btn btn-danger ml-4" onclick="return confirm('Biztosan szeretné véglegesen törölni a bejegyzését? Ez a művelet nem vonható vissza.')">Végleges törlés</button>

web.php-n adjuk meg az új útvonalat:

Route::delete('/trashed/{note}', [TrashedNoteController::class, 'destroy'])->withTrashed()->middleware('auth')->name('trashed.destroy');

TrashedNoteController tartalmát is ki kell egészíteni:

public function destroy(Note $note){

if(!$note->user->is(Auth::user())){

return abort(403);

}

$note->forceDelete();

return to\_route('trashed.index', $note)->with('success', 'Feljegyzés véglegesen törölve');

}

Ezzel el is készült az első alkalmazásunk. ☺