Ruzsinszki Zita

2024.11.

Lite Notes

Laravel11

Tartalom

[1. Specifikáció 3](#_Toc182421638)

[2. Készítsünk új projektet: 5](#_Toc182421639)

[3. Breeze és Jetstream 5](#_Toc182421640)

[4. Kezdő lap cseréje 7](#_Toc182421641)

[5. Title átírása 9](#_Toc182421642)

[6. Első migráció végrehajtása 9](#_Toc182421643)

[7. Egyedi útvonalak megadása 10](#_Toc182421644)

[8. Bejelentkezés utáni átirányítás testreszabása 12](#_Toc182421645)

[9. Bejegyzések beolvasása 13](#_Toc182421646)

[10. A Felhasználó Jegyzeteinek Lekérése az Eloquent Segítségével 14](#_Toc182421647)

[11. Jegyzetek renderelése a böngészőben 16](#_Toc182421648)

[12. A @forelse és @empty direktívák 19](#_Toc182421649)

[13. Lapozó hozzáadása 20](#_Toc182421650)

[14. Text limit 21](#_Toc182421651)

[15. Új feljegyzés hozzáadása a weboldalról 23](#_Toc182421652)

[16. Űrlapon felvitt adatok mentése az adatbázisba 26](#_Toc182421653)

[16.1 .Komponensek használata a hibakezeléshez 27](#_Toc182421654)

[16.2. Az old direktíva használata az űrlapban 28](#_Toc182421655)

[16.3. Hibakezelés a Kontrollerben 29](#_Toc182421656)

[17. Tömeges Hozzárendelés Védelme az Eloquent Modellekben 30](#_Toc182421657)

[18. Egyedi azonosítók 31](#_Toc182421658)

[19. Bejegyzések szerkesztése 34](#_Toc182421659)

[20. Bejegyzések törlése 37](#_Toc182421660)

[21. Visszajelzések a usernek a végrehajtott műveletek sikerességéről 40](#_Toc182421661)

[21.1. Értesítés a módosítás sikerességéről 40](#_Toc182421662)

[21.2. Értesítés a törlés sikerességéről 42](#_Toc182421663)

[22. Eloquent kapcsolatok 42](#_Toc182421664)

[22.1. Egy az Egyhez (One-to-One) 43](#_Toc182421665)

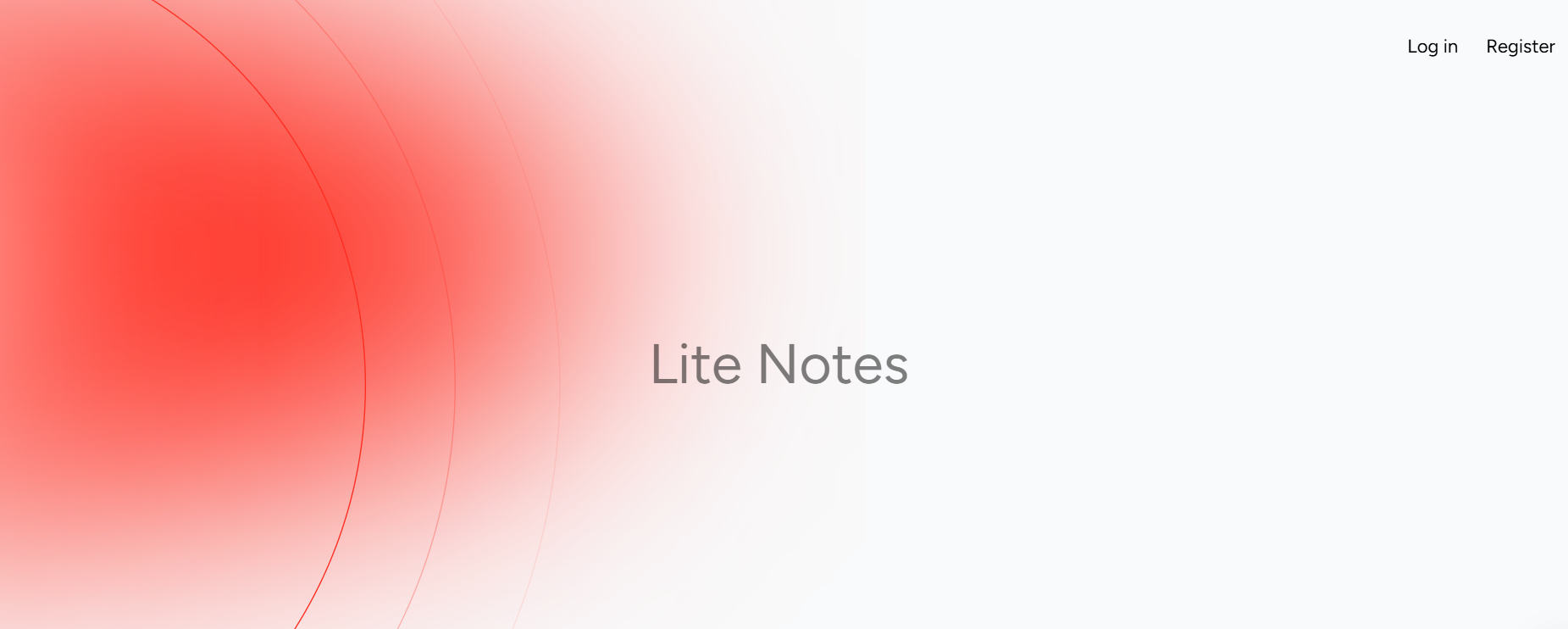
[22.2. Egy a Sokhoz (One-to-Many) 43](#_Toc182421666)

[22.3. Sok a Sokhoz (Many-to-Many) 44](#_Toc182421667)

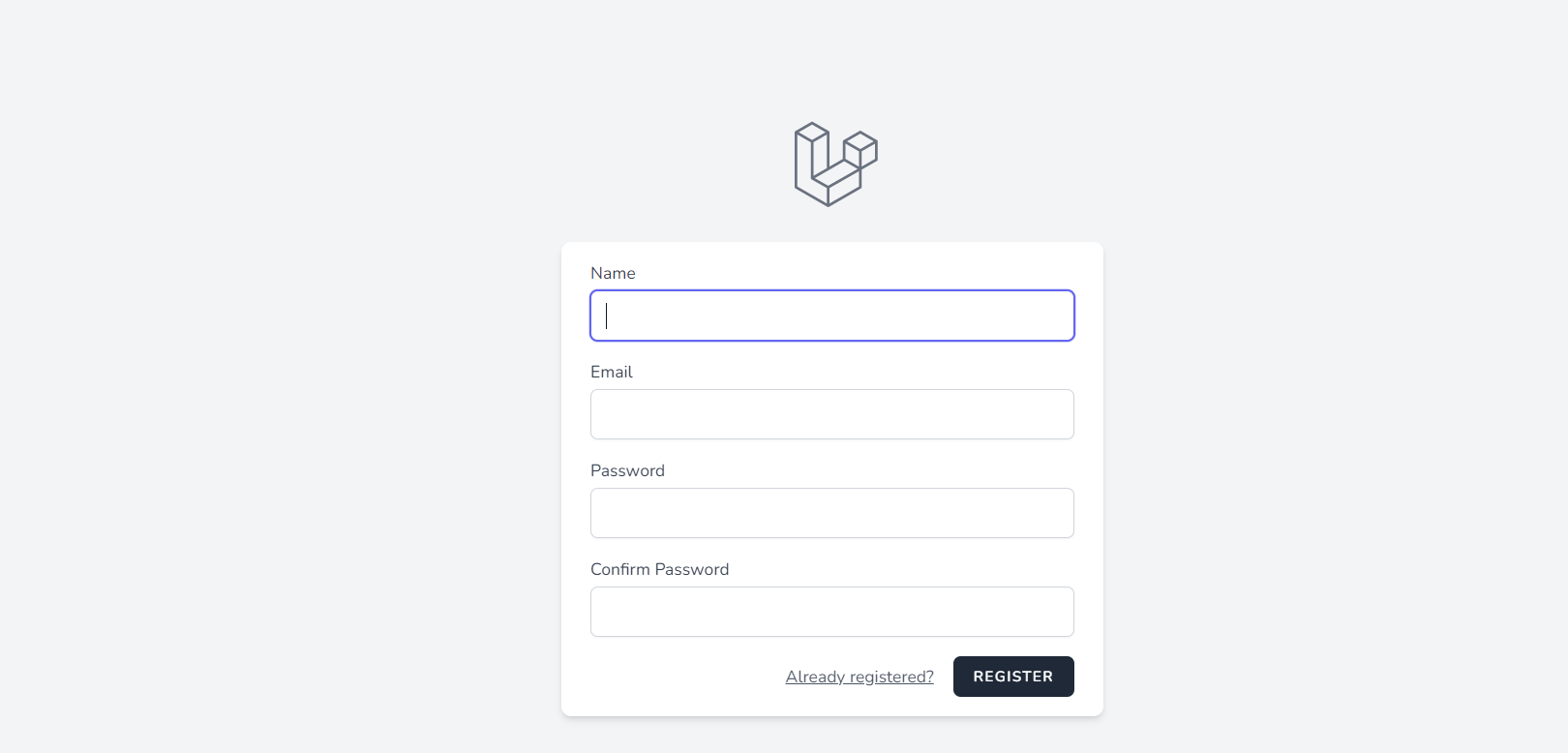
[22.4. Egy a Sok Közül (Has Many Through) 44](#_Toc182421668)

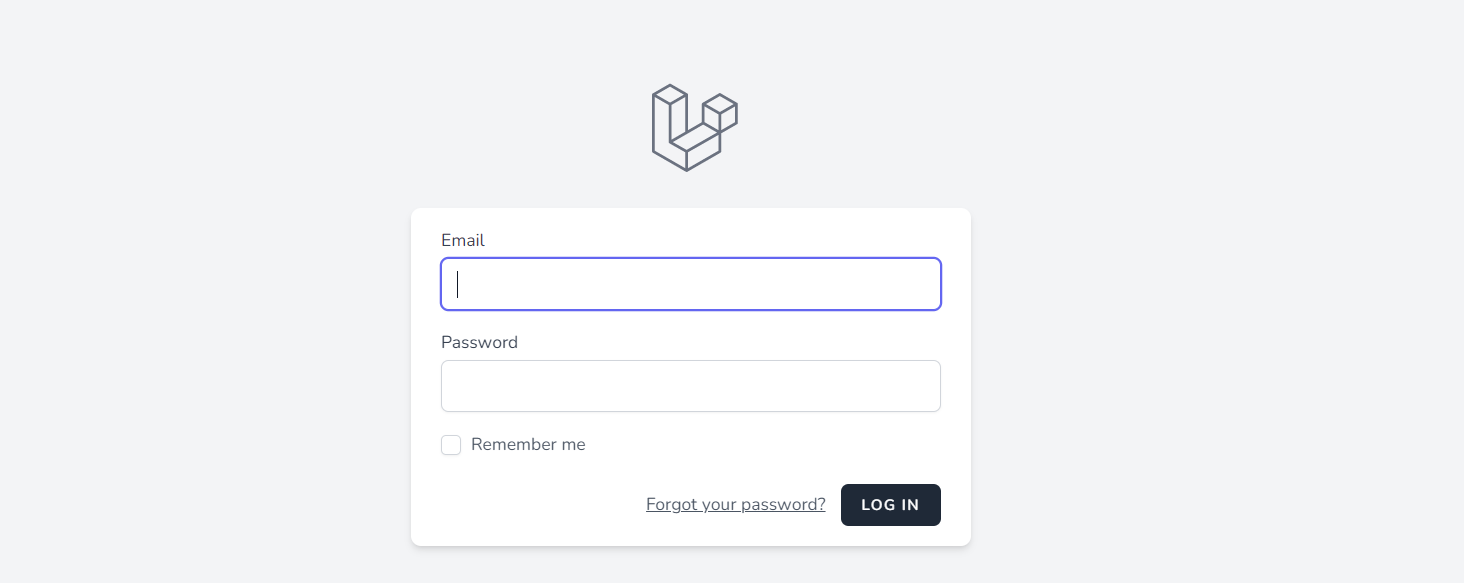
# Specifikáció

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

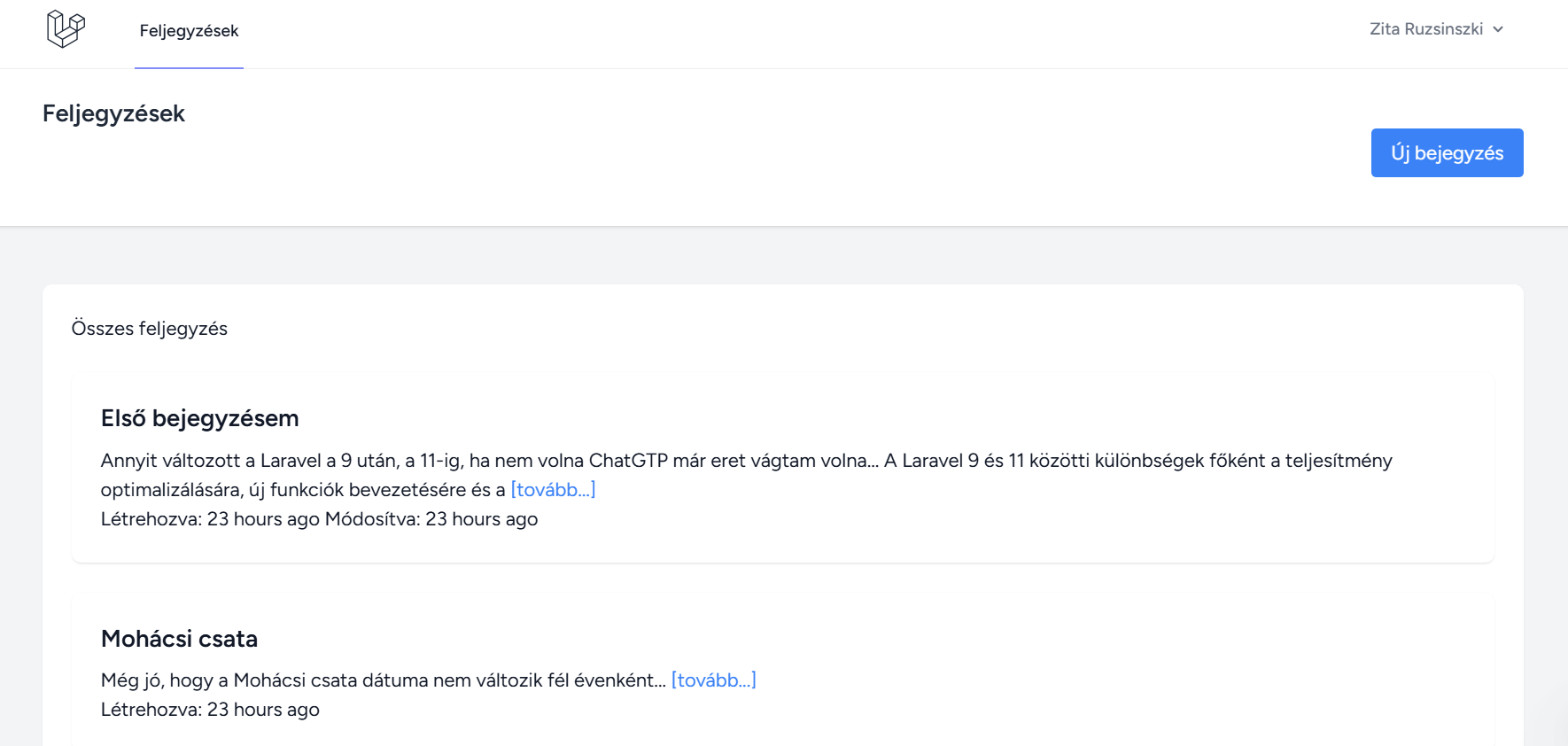


Az új felhasználónak névvel, e-mail címmel és jelszóval kell regisztrálnia. Ez létrehoz egy új fiókot, és bejelentkezteti a felhasználót.

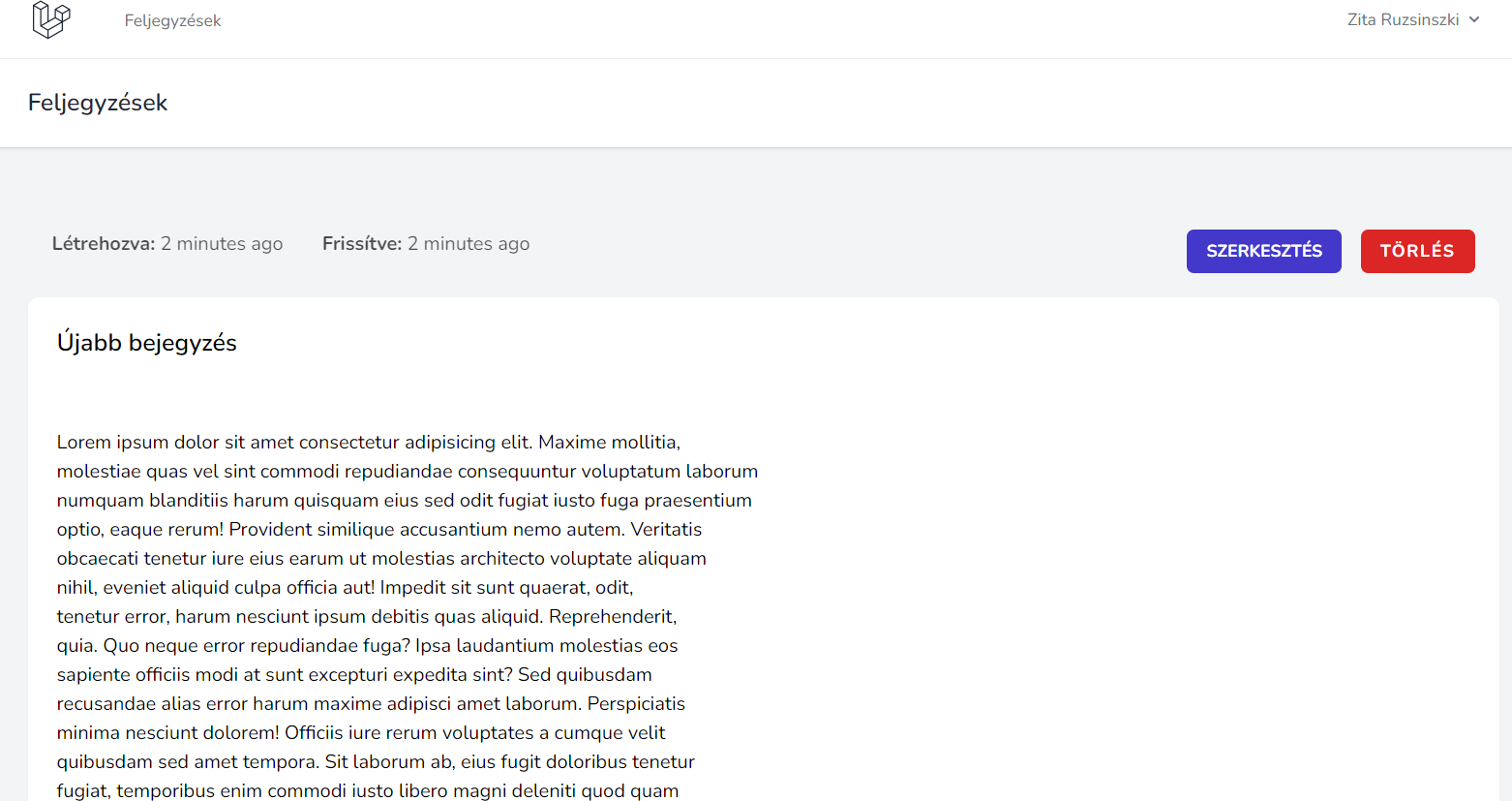
 Egy meglévő felhasználó bejelentkezhet a bejelentkezési oldalról.



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:

Telepítsünk egy új projektet: laravel new LiteNotes

Állítsuk be a .env fájlt:

DB\_CONNECTION=mysql

DB\_HOST=localhost

DB\_PORT=3306

DB\_DATABASE=litenotes

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.

Telepítés lépéseit a dokumentációban is megtalálod: <https://laravel.com/docs/11.x/starter-kits#breeze-and-blade>

composer require laravel/breeze –dev

Miután a Composer telepítette a Laravel Breeze csomagot, futtatni kell a breeze:install Artisan parancsot. Ez a parancs közzéteszi az authentikációhoz tartozó nézeteket, útvonalakat, vezérlőket és egyéb erőforrásokat az alkalmazásodban. A Laravel Breeze az összes kódját publikálja az alkalmazásodba, így teljes kontrollod és rálátásod lesz a funkciókra és a megvalósításra.

A breeze:install parancs megkérdezi majd, melyik frontend stack-et és tesztelési keretrendszert részesíted előnyben. Itt majd add meg a blade-et.

php artisan breeze:install

php artisan migrate

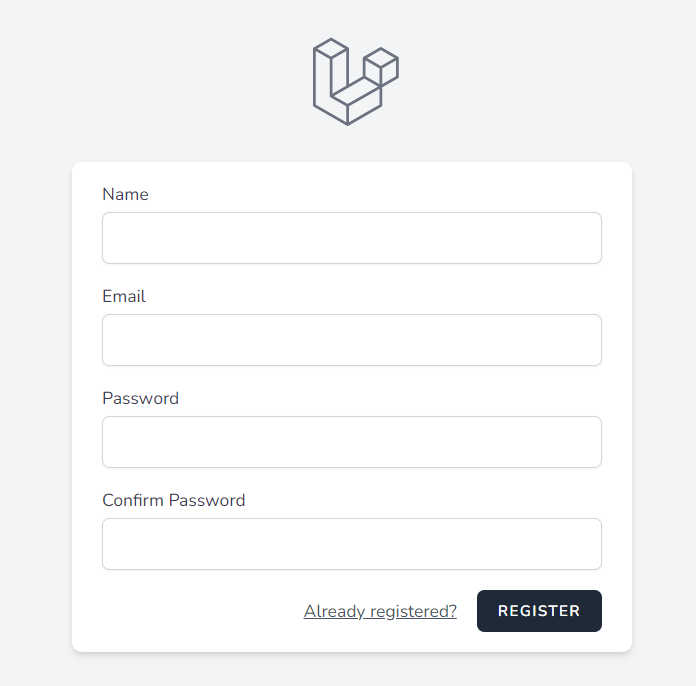
npm install

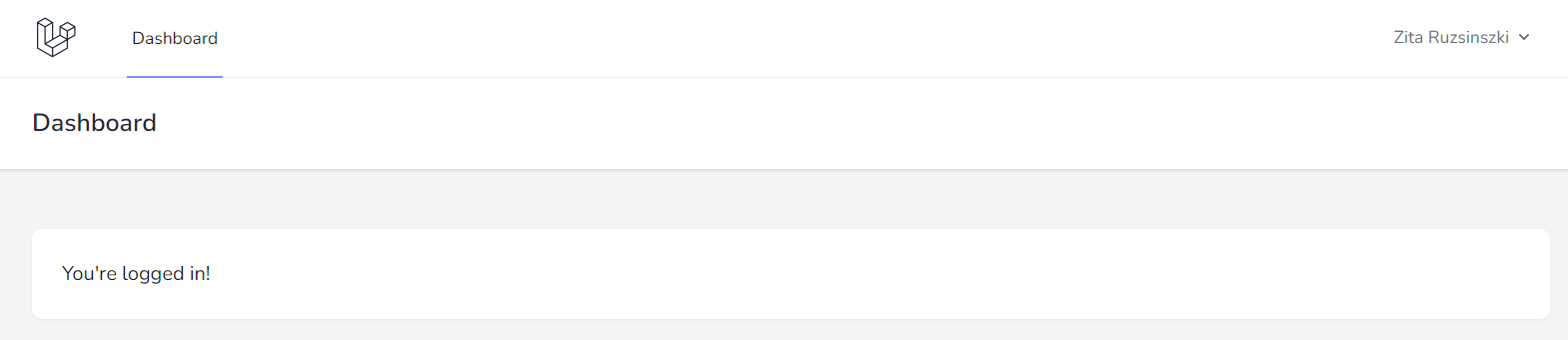
php artisan serve parancsot (*egy terminálon először ezt kell elindítanod, majd nyiss egy új terminált és ott a következővel indítsd el a frontend figyelését*)

npm run dev (*ezzel a frontend oldal indul csak, a php servert nem indítja el*)

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)

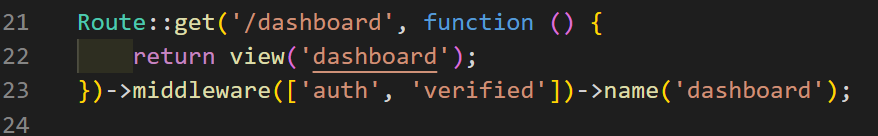
<http://127.0.0.1:8000/register> -> itt már ki is próbálhatod a regisztrációt

 Remélhetőleg ezt látod, ha igen töltsd ki a regisztrációt, 8 karakter hosszúnak kell lennie a jelszónak minimum.



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://127.0.0.1:8000/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 .env 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.

# 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.

Maradt némi formátum még így is. Keressük meg, honnan kapja meg ezt a stílust.Ha megnézed pl: a **resources\views\auth\login.blade.php**

itt az első sorban láthatod: **<x-guest-layout>** koponens került felhasználásra, ami a layout mappában található: **resources\views\layouts\guest.blade.php** itt megtaláljuk a css-re való hivatkozást.

A <x-guest-layout> komponens a Blade komponens rendszer része Laravelben, ami lehetővé teszi a kód újrafelhasználását és az oldalak felépítésének egyszerűsítését. Az <x-guest-layout> itt egy "vendég" layout, amely olyan oldalakhoz használatos, ahol a felhasználó nincs bejelentkezve (pl. bejelentkezési vagy regisztrációs oldalak).

A guest.blade.php layout fájl tartalmazza a HTML struktúrát és a szükséges CSS vagy JS hivatkozásokat, amik minden olyan oldalon megjelennek, ahol a <x-guest-layout> komponenst használjuk. Ez azt jelenti, hogy az egyes aloldalakra (pl. login) már nem kell külön hozzáadni ezeket a hivatkozásokat, mivel a guest.blade.php layoutban egyszer kerültek meghatározásra.

Ha megnézed a head-ben már szerepel a hivatkozás feltételesen egy elágazásban.

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

A Header osztályait javítottam:

<header class="fixed top-0 right-0 p-6 z-10">

A main elemek közé, köré adjuk hozzá:

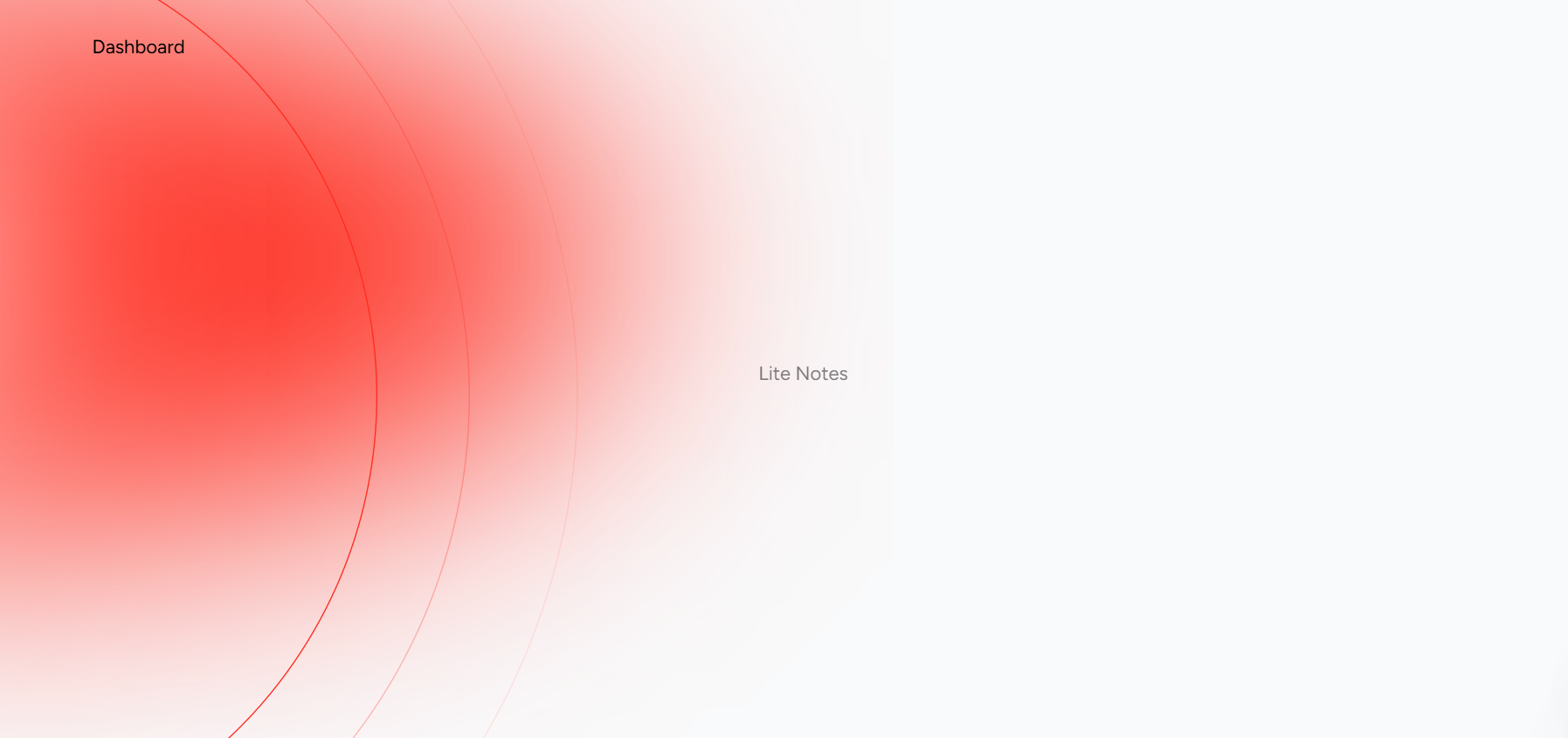
<div class="flex-grow flex items-center justify-center">

<main class="text-center">

<h1 class="text-5xl">Lite Notes</h1>

</main>

</div>



welcome.blade.php

A public\build\assets\app-Bdhvm1QU.css helyen találod a css-t. A saját css-t is a public mappán belül kell hozzá adni. Hozz létre a public mappán belül egy css mappát, azon belül egy mycss.css fájlt. Csatold a welcome oldalhoz, és adj meg a h1 elemnek kb 5em betű méretet.

A resources\css\app.cssa tailwind, de a vsc azt írja unknown rules, megoldás: (de működik e nélkül is)

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

@import 'tailwindcss/base';

@import 'tailwindcss/components';

@import 'tailwindcss/utilities';

# Title átírása

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

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

Azért jó így megadni, mert később ha módosítod az .env fájlban az APP\_NAME értékét, például "My New App"-re, akkor a változás automatikusan tükröződik az alkalmazás címében anélkül, hogy minden egyes fájlt egyenként kellene frissítened. Ha szóközzel akarod megadni az APP\_NAME értékét, akkor tedd idézőjelbe. (próbáld ki anélkül is)

# Első migráció végrehajtása

Most készítsük el az első modellünket a migrációval együtt, a jegyzet készítő alkalmazásunkhoz.

Tehát szükségünk van egy jegyzet táblázatra és egy megfelelő jegyzet modellre. A terminálon adjuk ki a következő parancsot:

php artisan make:model Note –m

(A Laravel 11 érzékenyebb, a –m előtt javítsd át a terminálon a kötőjelet, mert innen duplán fogja venni, és hibaüzenetet kapsz)

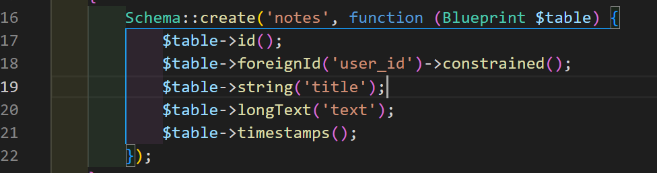
A modellnévnek nagybetűvel kell kezdődnie, és egyes számban kell lennie, ezért Note és nem notes. Az -m opció, automatikusan létrehozza a hozzátartozó migrációs fájlt, vagyis a táblát.

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 phpmyadmin-t és nézzük meg létrehozta e a táblát. Ha igen, akkor készen van a Note modellünk is.

# Egyedi útvonalak megadása

A web.php fájlban a következő útvonalak mindegyike külön-külön biztosítja az alkalmazás különféle funkcióit:

* Route::get('/notes', );: A notes lista megjelenítéséhez szükséges útvonal. Ez általában egy indexoldalt szolgál ki, ahol az összes Note típusú adatot megjelenítjük.
* Route::get('/notes/{note}', );: Egy konkrét Note megtekintéséhez használatos, ahol {note} az adott Note ID-ját jelöli.
* Route::get('/notes/create', );: Új Note létrehozására szolgáló űrlap megjelenítéséhez vezető útvonal.
* Route::post('/notes', );: Az új Note űrlap elküldésére használt útvonal, amely létrehozza az új adatot.
* Route::get('/notes/{note}/edit', );: Egy adott Note szerkesztésére szolgáló űrlap megjelenítésére használatos.
* Route::put('/notes/{note}', );: A szerkesztett Note mentésére használt útvonal, amely frissíti a meglévő adatot.
* Route::delete('/notes/{note}', );: Egy adott Note törlésére szolgáló útvonal.

Ezek az útvonalak biztosítják az összes alapvető CRUD (Create, Read, Update, Delete) művelet elvégzését egy Note típusú erőforráson.

**Route::resource használata**

Laravelben azonban van egy kényelmesebb és rövidebb mód ezeknek az útvonalaknak a meghatározására: a Route::resource használata. Ezzel az egyetlen sorral Laravel automatikusan létrehozza az összes szükséges útvonalat egy erőforráshoz, mint például a notes-hoz, a megfelelő HTTP metódusokkal (GET, POST, PUT, DELETE).

Route::resource('/notes', 'NoteController');

A Route::resource a következő útvonalakat generálja:

| **HTTP Metódus** | **Útvonal** | **Controller Művelet** | **Leírás** |
| --- | --- | --- | --- |
| GET | /notes | index | Az összes Note listázása |
| GET | /notes/create | create | Új Note létrehozási űrlap |
| POST | /notes | store | Új Note elmentése az adatbázisba |
| GET | /notes/{note} | show | Egy konkrét Note megjelenítése |
| GET | /notes/{note}/edit | edit | Egy Note szerkesztési űrlap |
| PUT/PATCH | /notes/{note} | update | Egy Note frissítése |
| DELETE | /notes/{note} | destroy | Egy Note törlése |

Ezeket a metódusokat a NoteController-ben kell definiálni, hogy az egyes útvonalak megfelelően működjenek.

Tehát, a Route::resource('/notes', 'NoteController') használata egyszerűsíti az útvonalak definiálását és biztosítja az összes alapvető CRUD funkciót egy erőforráshoz, ami könnyebben karbantartható és olvasható kódot eredményez.

<https://laravel.com/docs/11.x/eloquent-resources#concept-overview>

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

Ez a parancs létrehozza a NoteController.php fájlt a app/Http/Controllers könyvtárban, amely tartalmazza az összes szükséges metódust a CRUD műveletekhez.

A parancs hatására a következő alapértelmezett metódusok generálódnak a NoteController-ben:

* index: Az összes Note megjelenítése.
* create: Új Note létrehozási űrlap megjelenítése.
* store: Az új Note mentése az adatbázisba.
* show: Egy konkrét Note megjelenítése.
* edit: Egy Note szerkesztési űrlap megjelenítése.
* update: Egy meglévő Note frissítése.
* destroy: Egy Note törlése.

Ha végiggörgetünk a fájlon, láható, hogy minden szükséges metódus létrejött.

Most térjünk vissza a web.php fájlra és mostmár meg tudjuk adni a második paramétert is:

use App\Http\Controllers\NoteController;

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

 **Route::resource**: Ez létrehozza az összes CRUD útvonalat a notes erőforráshoz (index, create, store, show, edit, update, destroy).

 **NoteController::class**: Itt az App\Http\Controllers\NoteController kontrollert használja az összes notes művelet kezelésére.

 **->middleware(['auth'])**: Ez biztosítja, hogy az útvonalakhoz tartozó összes művelethez szükség legyen hitelesítésre, azaz csak bejelentkezett felhasználók érhetik el őket.

# Bejelentkezés utáni átirányítás testreszabása

Nyisd meg az app/Http/Controllers/Auth/AuthenticatedSessionController.php fájlt és a store metódusban írd át a visszatérési értéket a következő értékre:

return redirect()->intended('/notes');

Ezzel a sorral közvetlenül megadjuk, hogy a bejelentkezés után a felhasználókat a /notes oldalra irányítsa.

Működése:

A redirect() egy beépített Laravel függvény, amely átirányít egy megadott URL-re vagy útvonalra.

A intended() metódus arra szolgál, hogy a felhasználót az eredetileg tervezett oldalra irányítsa bejelentkezés után, ha egy védett (hitelesítést igénylő) oldalra próbált belépni. Ha a felhasználó nem próbált másik céloldalt elérni, akkor a megadott /notes útvonalra kerül.

**Ha a felhasználó egy másik oldalra próbált eljutni**: Például ha közvetlenül a /notes/1/edit oldalra próbálna belépni bejelentkezés nélkül, akkor a rendszer a bejelentkezési oldalra irányítja. Miután sikeresen bejelentkezik, a intended() metódus felismeri az eredeti célt, és visszairányítja a felhasználót a /notes/1/edit oldalra, nem pedig a /notes-ra.

**Ha nincs más cél beállítva**: Ha a felhasználó egyszerűen a bejelentkezési oldalon keresztül lép be, és nem próbált védett oldalra eljutni, akkor a intended('/notes') metódus automatikusan a /notes oldalra irányítja.

# Bejegyzések beolvasása

A jegyzetoldalunk most még üres. Viszont ha lesznek jegyzetek azt a NoteController segítségével fogjuk tudni megjeleníteni.

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

INSERT INTO `notes` (`id`, `user\_id`, `title`, `text`, `created\_at`, `updated\_at`) VALUES ('1', '1', 'Első bejegyzésem', 'Annyit változott a Laravel a 9 után, a 11-ig, ha nem volna ChatGTP már eret vágtam volna...', NULL, NULL);

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;

# A Felhasználó Jegyzeteinek Lekérése az Eloquent Segítségével

Az Eloquent ORM-et használva így kérhetjük le a bejelentkezett felhasználó összes jegyzetét. Nyisd meg a NoteControllert és egészítsd ki az index metódust a következő sorokkal: 

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

Fontos:

use App\Models\Note;

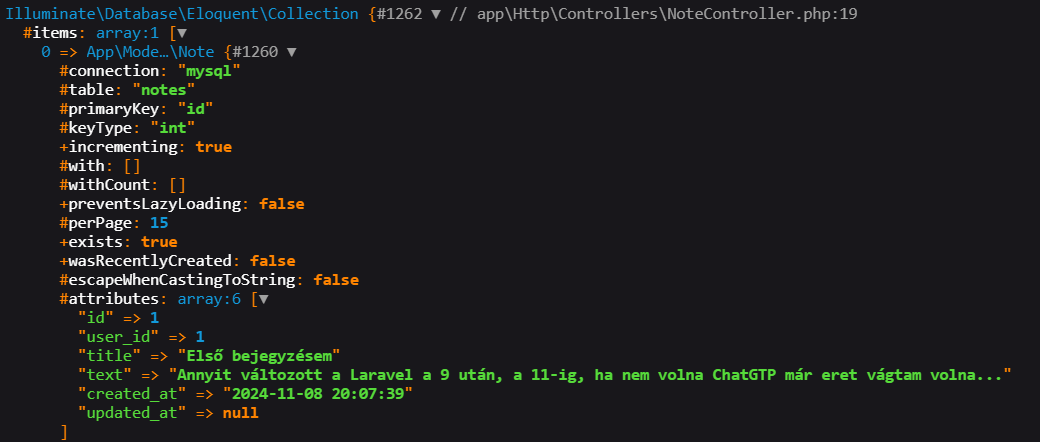
 Note::where('user\_id', $userId)->get(); egy Eloquent lekérdezés, amely azt mondja a rendszernek, hogy csak azokat a jegyzeteket kérje le, ahol a user\_id megegyezik a bejelentkezett felhasználó azonosítójával.

 A get() metódus egy tömböt ad vissza az összes egyező jegyzetről.

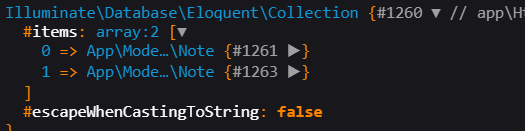
dd($notes);

A dd($notes); segítségével az összes lekérdezett jegyzetet megjelenítheted, hogy ellenőrizd, helyesen működik-e a lekérdezés. A dd (dump and die) egy Laravel függvény, amely kiírja a változó tartalmát és leállítja a kód futását, így teszteléshez nagyon hasznos.

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



Készítsünk még egy jegyzetet, hasonlóan, mint az előbb a phpmyadmin felületen. 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();

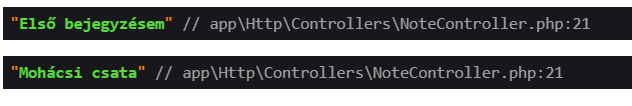
$notes->each(function($note){

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();

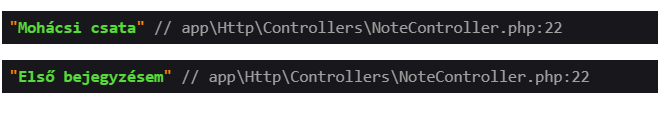
$notes->each(function($note){

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.

kommentezd ki a teszteket

# 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,

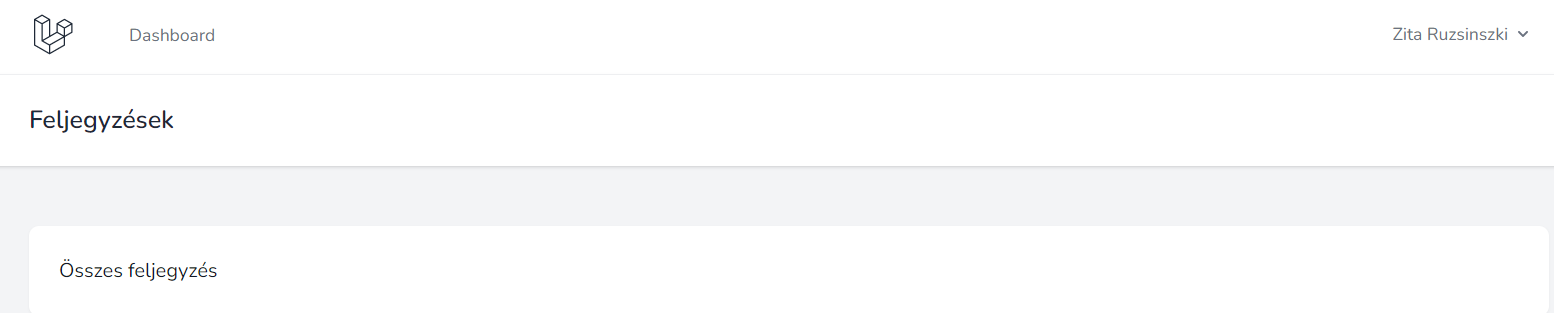
a 4ik sorban (dashboard helyett): {{ \_\_('Feljegyzések') }}

és 12-ik sorban (you are logged in helyett): {{ \_\_("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);**

 A view('notes.index') meghívja a resources/views/notes/index.blade.php nézetet.

 A with('notes', $notes) metódus hozzáadja a notes változót a nézethez, amely a $notes gyűjtemény értékét tartalmazza.

A with() metódus egy hasznos Laravel helper, amely lehetővé teszi, hogy adatokat adj át a nézeteknek (views). Ez egy kényelmes módja annak, hogy a nézetbe (view) változókat és azok értékeit adjuk át, így a nézetben hozzáférhetőek lesznek az adatok.

A with() metódus segítségével egy kulcs-érték pár formájában adunk át adatot. A kulcs (notes) a nézetben használandó változó neve lesz, az érték pedig az átadott adat, amit meg szeretnénk jeleníteni (ebben az esetben a $notes gyűjtemény).

A with() használata opcionális, mivel Laravelben többféle módon is átadhatsz adatokat a nézeteknek.

A with() nélkül is átadhatsz adatokat egy tömbként: return view('notes.index', ['notes' => $notes]);

**Láncolt with() hívások több adat átadásához**: Ha több változót is szeretnél átadni, akkor láncolhatod a with() hívásokat:

return view('notes.index')

->with('notes', $notes)

->with('title', 'Felhasználói Jegyzetek');

Összességében véve érdemesebb a with() –et használni, mert olvashatóbb, rugalmasabb és rövidebb lesz a kódod.

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

A {{ \_\_("Összes feljegyzés") }} sor kiírja a "Összes feljegyzés" szöveget a böngészőben. Az \_\_() függvény fordításokat tesz lehetővé, ami azt jelenti, hogy ha többnyelvű alkalmazást fejlesztünk, akkor a szöveg a nyelvi fájlok alapján automatikusan fordításra kerülhet.

Ez a Blade @foreach direktíva, amely egy ciklust indít a $notes gyűjtemény minden egyes elemén. A $notes gyűjtemény minden eleme (amely egy-egy jegyzet) egy $note nevű változóban kerül tárolásra a ciklus minden iterációjában.

A class attribútumban meghatározott CSS osztályok (pl. my-6, p-6, stb.) Tailwind CSS stílusokat használnak, amelyeket Laravel projektekben gyakran használnak az egyszerű stílusformázáshoz.

Ezek a stílusok a következőket jelentik:

* my-6: margin-y (felső és alsó margó) 6 egység.
* p-6: padding (belső margó) 6 egység minden oldalon.
* bg-white: fehér háttérszín.
* border-b border-gray-200: alsó szürke keret a border-gray-200 színkóddal.
* shadow-sm: kisebb árnyék a doboz körül.
* sm:rounded-lg: kis képernyőméreteknél lekerekített sarkak

A <h2> tag a jegyzet címét jeleníti meg. A {{ $note->title }} a jegyzet címét írja ki, amit az adatbázisból töltünk be.

A <p> tag a jegyzet szövegét jeleníti meg. A {{ $note->text }} az adatbázisból betöltött jegyzet szövegét írja ki.

Az @endforeach lezárja a @foreach ciklust. A ciklus minden egyes jegyzetet ($note) külön div-ben jelenít meg, a fent meghatározott formázással.

Így már a böngészőben szépen kirajzolódnak a bejegyzések:

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

Ha a jegyzetek létrehozási dátumát szeretnéd megjeleníteni a módosítási dátum helyett, akkor az updated\_at mező helyett a created\_at mezőt kell használni. Laravel automatikusan kezeli a created\_at és updated\_at mezőket, ha az timestamps opció engedélyezve van a modellben.

<span>Létrehozva: {{ $note->created\_at ? $note->created\_at->diffForHumans() : 'N/A' }}</span>

@if ($note->updated\_at && $note->updated\_at != $note->created\_at)

<span>Módosítva: {{ $note->updated\_at ? $note->updated\_at->diffForHumans() : 'N/A' }}</span>

@endifEllenőrizd, hogy a created\_at és updated\_at mezők léteznek-e és nem null értékűek, mielőtt formázni próbálod őket. Így elkerülhető az a hiba, amely akkor jelentkezik, ha valamelyik mező értéke null.

A dátum helyett egy felhasználó barátabb módon szokták inkább megjeleníteni az időt.

A diffForHumans() helper metódus a Laravelben található Carbon könyvtár része, amely egyszerű és olvasható formában jeleníti meg az időpontokat. Ahelyett, hogy pontos dátumot és időpontot adna vissza (például 2024-11-08 14:30:00), a diffForHumans() relatív időtartamot mutat, mint például „2 órája”, „3 nappal ezelőtt” vagy „1 hete”.

A diffForHumans() úgy számítja ki az időeltolódást, hogy az aktuális időhöz vagy egy másik időponthoz viszonyítja a dátumot, és relatív kifejezéssel adja vissza az eredményt.

# A @forelse és @empty direktívák

A @forelse és @empty direktívák használata a Blade-ben egy praktikus módszer a Laravelben tömbök (gyűjtemények) kezelésére, különösen akkor, ha az adott tömb esetleg üres. A @forelse ciklus úgy működik, mint a @foreach, de egy extra lehetőséget biztosít arra az esetre, ha a tömb üres lenne. Ez egy beépített, könnyen olvasható megoldás az üres adathalmazok kezelésére.

A @forelse ciklus a @foreach-hoz hasonlóan iterál egy tömb elemein, de tartalmaz egy @empty szakaszt is, amely akkor aktiválódik, ha a tömb üres.

**@forelse direktíva**:

* A @forelse ugyanúgy működik, mint a @foreach, végigmegy a $notes gyűjteményen.
* Minden note elemhez létrehoz egy div-et, amely megjeleníti a jegyzet címét, szövegét, és a létrehozás (valamint, ha módosítva lett, a módosítás) dátumát.

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**

 A @empty akkor fut le, ha a $notes gyűjtemény üres.

 Ilyenkor megjelenik egy üzenet, amely tájékoztatja a felhasználót, hogy nincsenek elérhető jegyzetek.

 Az @empty segítségével így elkerülheted, hogy az üres állapotokat külön logikával kelljen kezelni, mivel a @forelse automatikusan lefedi ezt az esetet.

A @endforelse zárja le a ciklust. A ciklusban mindig van egy @empty blokk, így biztos lehetsz benne, hogy akkor is történik valamilyen megjelenítés, ha a gyűjtemény üres.

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

# Lapozó hozzáadása

A laravelben 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);**

A paginate() metódus automatikusan kezeli a lapozást az adatbázis lekérdezés szintjén, így csak az aktuális oldalhoz szükséges adatokat kérdezi le.

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

az **index.blade.php-t** is egészítsük ki

@endforelse

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

</div>

 A {{ $notes->links() }} sor a lapozó navigációs elemeket jeleníti meg. Ezzel lehetőséget ad a felhasználónak, hogy az előző és a következő oldalra lépjen.

 Alapértelmezés szerint a links() Bootstrap-kompatibilis, így ha a Bootstrap könyvtárat használod, a lapozás stílusosan jelenik meg. Egyedi CSS-t is készíthetsz a saját lapozási stílusodhoz.

Ha most ráfrissítünk meg is jelenik a lapozó. Emlékeztek mennyit vacakoltunk vele az elején a Névjegyzék feladatban…



Adjunk meg az 1 helyett limitnek most 2-t, meg is fog jelenni mindkét bejegyzés.

# Text limit

A következő probléma, amit meg kell oldani, mi történjen akkor, ha egy bejegyzés túl hosszú.

A Laravel Str::limit() metódusa egy praktikus eszköz, amely segít abban, hogy hosszú szövegeket egy meghatározott karakterhosszra korlátozzunk, és egy rövidített verziót jelenítsünk meg. Ez különösen hasznos olyan helyzetekben, amikor például bejegyzések előnézeti részét szeretnénk megjeleníteni, és nem akarjuk az egész szöveget kiírni.

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>

 A {{ Str::limit($note->text, 200) }} csak az első 200 karaktert jeleníti meg a jegyzet szövegéből ($note->text).

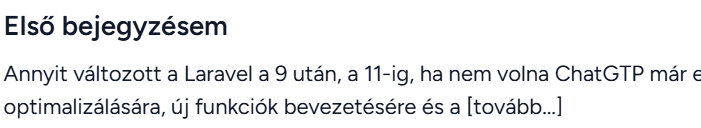
 Ha a szöveg hossza meghaladja a 200 karaktert, akkor a Laravel automatikusan hozzáad egy ... karakterláncot a végére, hogy jelezze, hogy van további tartalom.

A Str::limit() harmadik paraméterként elfogad egy egyéni végződést is. Ha más jelölést szeretnél használni a folytatás jelzésére, megadhatod azt is:

<p class="mt-2">{{ Str::limit($note->text, 200, ' [tovább...]') }}</p>

Itt a 200 karakter után a [tovább...] szöveget jelenítjük meg az alapértelmezett ... helyett.

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



Így, hogy sikeresen eltüntettük a 201-ik karaktertől kezdve a szöveget, valamilyen eseményre vissza kellene varázsolni.

Először is, alakítsuk linkké a „tovább…” szöveget, és tegyük annak látszóvá a megfejelő osztályok hozzáadásával:

<p class="mt-2">

{{ Str::limit($note->text, 200, ' ') }}

<a href="{{ route('notes.show', $note->id) }}" class="text-blue-500 hover:underline">[tovább...]</a>

</p>

Hozzunk létre egy új útvonalat a web.php fájlban a részletező fájlnak:

Route::get('/notes/{note}', [NoteController::class, 'show'])->name('notes.show');

Ez az útvonal a NoteController show metódusához vezet, és egy adott jegyzet (note) azonosítóját (id) fogadja paraméterként.

Következő lépésként készítsünk egy új show metódust a NoteController-ben, amely egyedi bejegyzést tölt be, és továbbítja azt a részletező nézetnek (show.blade.php). Magát a metódust legenárlja a laravel, csak ki kell töltened:

// Show metódus a teljes bejegyzés megjelenítéséhez

public function show(Note $note)

{

// Ellenőrizzük, hogy a felhasználó valóban hozzáférhet-e a jegyzethez

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

abort(403); // Hozzáférés megtagadása, ha a jegyzet nem a bejelentkezett felhasználóé

}

return view('notes.show', compact('note'));

}

Majd írjuk meg a show.blade.php fájlunkat is:

@extends('layouts.app')

@section('content')

<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> <!-- Teljes szöveg megjelenítése -->

<span>Létrehozva: {{ $note->created\_at ? $note->created\_at->diffForHumans() : 'N/A' }}</span>

@if ($note->updated\_at && $note->updated\_at != $note->created\_at)

<span>Módosítva: {{ $note->updated\_at ? $note->updated\_at->diffForHumans() : 'N/A' }}</span>

@endif </div>

<!-- Vissza link a főoldalra -->

<a href="{{ route('notes.index') }}" class="text-blue-500 hover:underline">Vissza a jegyzetekhez</a>

@endsection

**Bejegyzések címének linkké alakítása**

Nem árt, ha a címre kattintva is eljutunk a teljes cikkhez. Egészítsük ki, az index.blade.php oldalon a címet:

<h2 class="font-semibold text-xl**"><a href="{{ route('notes.show', $note->id) }}"> {{ $note->title }} </a>**</h2>

**Dashboard eltüntetése**

A weboldal tetején még mindig a Dashboard szöveg szerepel, ezt is 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, vagy inkább hagy ott. ☺

# 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:

<div class="flex justify-end mb-4">

<a href="{{ route('notes.create') }}" class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600">

Új bejegyzés

</a>

</div>

Így a jobb oldalra fog kerülni, én a Feljegyzések alá illesztettem bele.

A css-t is kiegészíthetjük (ezt ne a public mappában lévő mycss-be, hanem a resoures css mappában a tailwind css-hez add hozzá. Alá fogja húzni a VSC, de attól működik)

@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, - amennyiben egy másik terminálon elindítottad az npm run dev paranccsal - és ha rákattintunk az url-nek is működnie kell. Igaz egyenlőre egy üres oldalt kapunk, de legalább hibát nem jelez.

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">

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

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

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

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

</form>

</div>

</div>

</div>

</div>

</x-app-layout>

 **w-full**: A mezők teljes szélességet kapnak a szülő konténeren belül.

 **px-4 py-2**: Vízszintes (px-4) és függőleges (py-2) belső margókat ad, hogy kényelmesen lehessen gépelni a mezőkbe.

 **border border-gray-300**: Szürke keretet adunk a mezőknek.

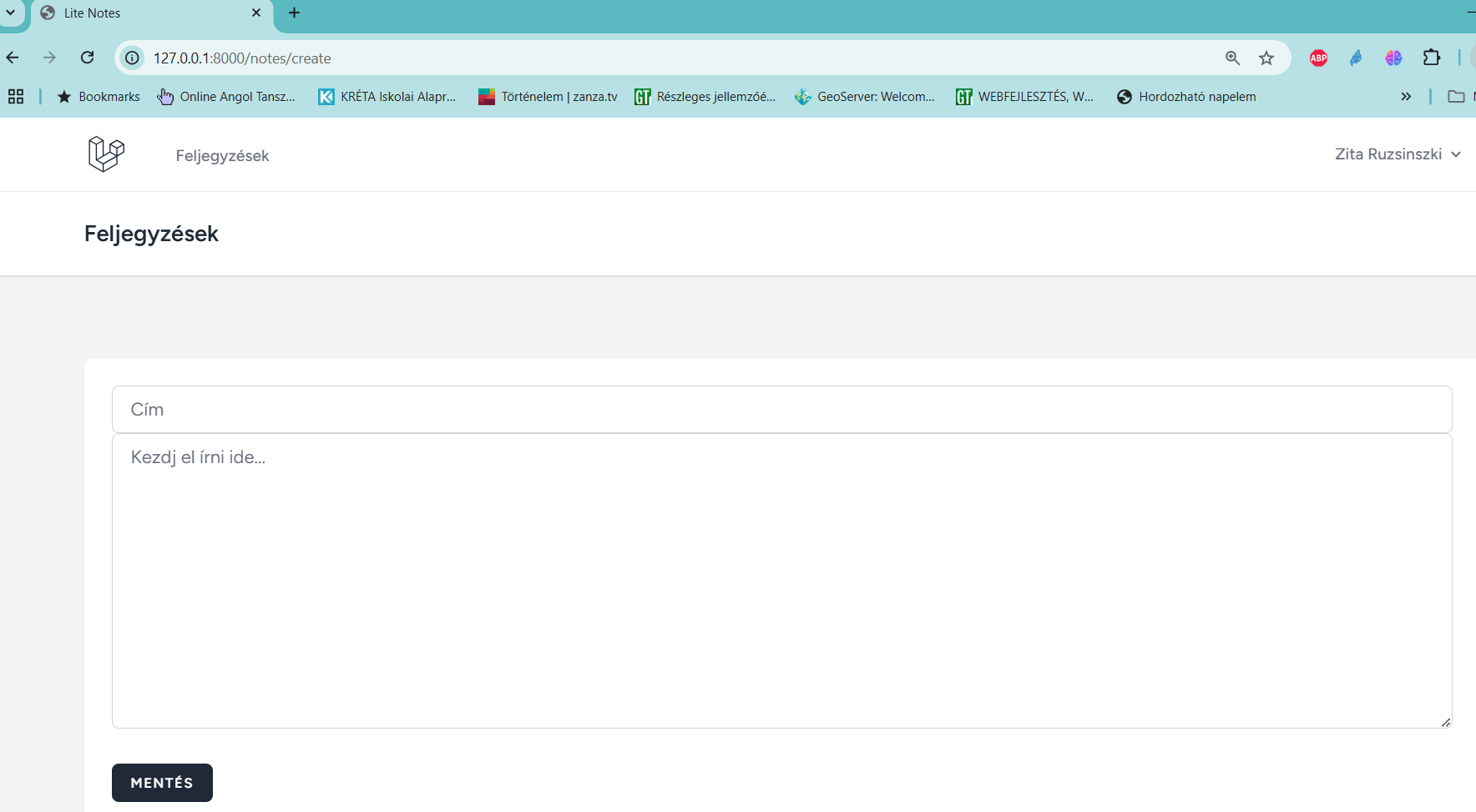
 **rounded-md**: A sarkok lekerekítéséhez.

 **focus:outline-none**: Eltávolítja az alapértelmezett fókusz körvonalat.

 **focus:ring-2 focus:ring-blue-500 focus:border-transparent**: Amikor a mező fókuszban van, kék színű keretet jelenít meg (focus:ring-2 és focus:ring-blue-500), és átlátszóvá teszi a border (focus:border-transparent).

 **resize-none** (csak a textarea mezőhöz): Megakadályozza, hogy a textarea átméretezhető legyen.

A components mappán belül található egy text-input.blade.php abból ki lehet puskázni a tailwind osztályait, amik használatban vannak, így az új fájlod azonnal megkapja a megfelelő formátumot.

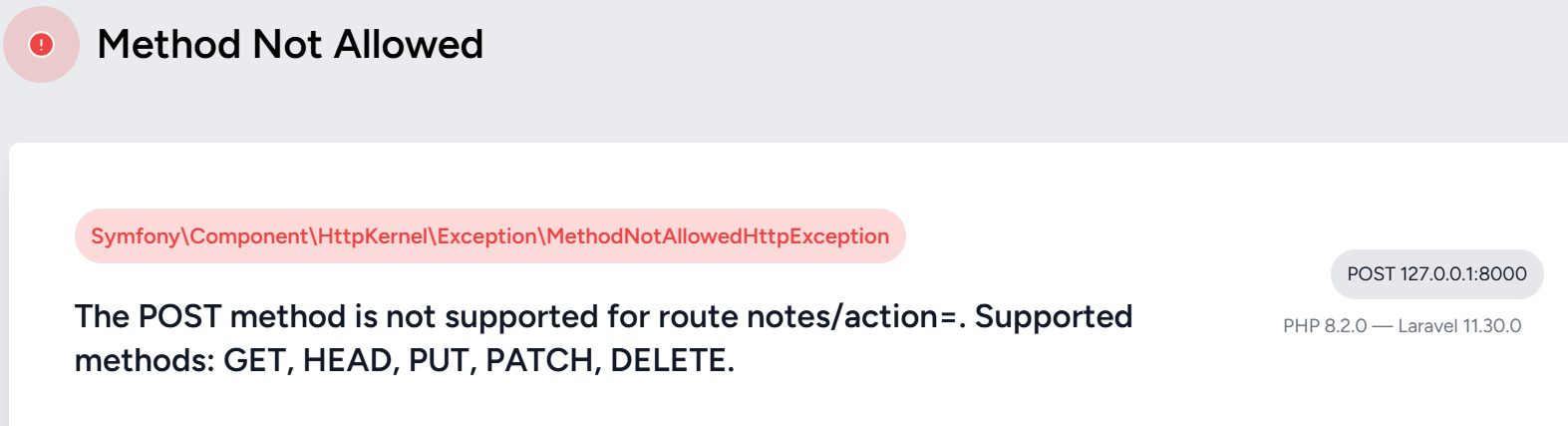


Az felvitt adatokat el is kell tárolnunk, így az action-t egészítsük ki:

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

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

Eredmény:



Ez a hibaüzenet azt jelzi, hogy a Laravel nem engedélyezi a POST metódust az adott útvonalhoz (notes/action=). Az ilyen hiba általában akkor fordul elő, ha az útvonal nincs megfelelően definiálva a web.php fájlban, vagy a POST metódus nincs engedélyezve az adott útvonalon.

https://laravel.com/docs/11.x/csrf#main-content

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() }}" />

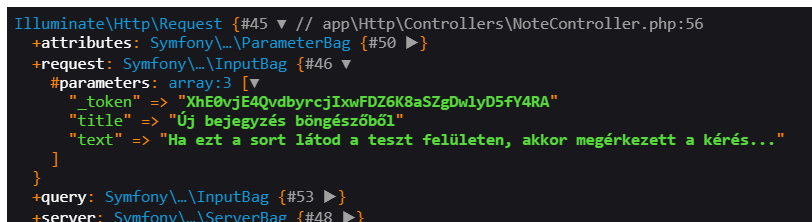
De nem kell minden alkalommal hozzáadni az egész elemet, egyszerűen használhatjuk a CSRF blade direktívát az űrlapon belül, így**:@csrf**. Adjuk hozzá az űrlaphoz az input mező elé, és máris megszűnik a hiba üzenet, ha megpróbálunk új bejegyzést hozzáadni és menteni próbáljuk, de mentésre még nem fog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

# 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 adatbázisba, fontos, hogy érvényesítsük őket, így biztosítva, hogy megfeleljenek az elvárt formátumnak és szabályoknak. Ezt az érvényesítést a kérés ($request) objektum által biztosított validate metódussal végezhetjük el, az alábbi módon:

$request->validate([

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

'text' => 'required|string',

]);

* **title mező**: Beállítjuk, hogy kötelező legyen (required), szöveg típusú (string), és legfeljebb 120 karakter hosszú lehet (max:120).
* **text mező**: Szintén kötelező (required) és szöveg típusú (string).

Az érvényesítési szabályokat kulcs-érték párok formájában adjuk meg, ahol a kulcs a mező neve (title, text), az érték pedig a hozzá tartozó érvényesítési szabályok listája. Ha az adatok megfelelnek ezeknek a szabályoknak, akkor az adatbázisba menthetők; ellenkező esetben a Laravel automatikusan visszajelzést ad a felhasználónak az érvénytelen mezőkről.

Az összes elérhető érvényesítési szabály megtalálható a laravel dokumentációban itt: <https://laravel.com/docs/11.x/validation>

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 az errors változóban érhetők el.

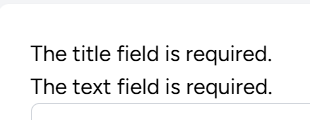
Nyissuk meg a **resources\views\notes\create.blade.php** fájlunkat, és adjuk hozzá, ha hibát talál a validálás során a program a form mezők fölé írja ki.

@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, töröld ki az előzőt és add hozzá így:

<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**

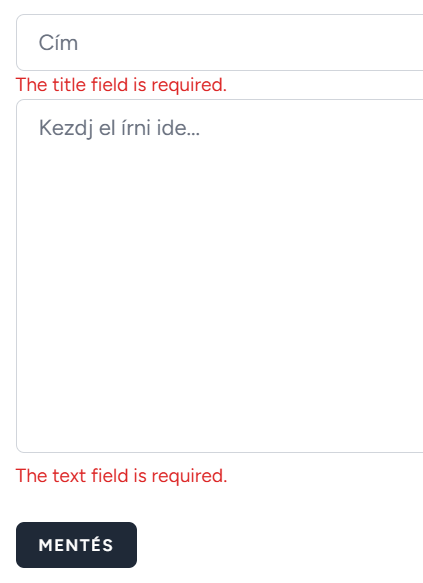
<x-textarea name="text" rows="10" placeholder="Kezdj el írni ide..." class="w-full mt-6">

</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.

Az @error direktíva lehetővé teszi számunkra, hogy az egyes mezők alá megjelenítsük a validációs hibákat. Amikor egy mező validációs szabályokat sért, a Laravel automatikusan visszaad egy hibaüzenetet, amelyet az @error segítségével egyszerűen megjeleníthetünk.

Ez is egy jó megoldás, de jobban ki lehet használni a Laravel adottságait, és a hibaüzeneteket áthelyezhetnénk a components-be. Szinte hallom, jóvanezmáritt ☺

A Laravel beépített validációs rendszere lehetővé teszi, hogy az űrlapunk mezőire érvényességi szabályokat alkalmazzunk, és a hibákat közvetlenül a felhasználó számára jelezzük. Az @error Blade direktíva segítségével a hibák megjelenítése egyszerű, miközben a @props direktíva segítségével dinamikusan kezelhetjük a komponensekben megjelenő mezőket.

## .Komponensek használata a hibakezeléshez

Ahelyett, hogy minden egyes űrlapmezőnél külön meghatároznánk az @error direktívát, létrehozhatunk különböző komponenseket az input és textarea mezőkhöz, így újrahasználhatjuk őket más űrlapokban is. Ezekben a komponensekben a @props segítségével állítjuk be a mezők jellemzőit, így dinamikusan átadhatunk tulajdonságokat.

Keressük meg a **components**ben azokat a fájlokat, amelyekre az x-slot-tal hivatkoztunk.

Az egyik, amit szerkesztenünk kell az már létezik az **input-text.blade-php** fájl. Egészítsd ki:

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

<input

name="{{ $field }}"

{{ $disabled ? 'disabled' : '' }}

{{ $attributes->merge(['class' => 'w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent']) }}

value="{{ old($field, $value) }}"

* //ezzel nem lesz jó, mert megduplázza a login input mezőit

@error($field)

<div class="text-red-600 text-sm mt-2"> {{ $message }} </div>

@enderror

A text-input.blade.php komponens létrehoz egy input mezőt a dinamikusan beállítható name, disabled, value és class attribútumokkal. Az old függvényt használjuk a value érték beállítására, hogy ha egy korábbi beküldés során hiba történt, az érték megmaradjon.

A textarea-nak hozz létre a components mappában egy új fájlt **textarea.blade.php**-t a következő tartalommal:

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

<textarea

name="{{ $field }}"

{{ $attributes->merge(['class' => 'w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent resize-none']) }}

rows="10"

>{{ old($field, $value) }}</textarea>

@error($field)

<div class="text-red-600 text-sm mt-2"> {{ $message }} </div>

@enderror

A textarea mező nem rendelkezik value attribútummal, ezért az értéket közvetlenül a nyitó és záró <textarea> címkék között adtuk meg. Az old() függvény segítségével biztosítjuk, hogy a mezőbe beírt tartalom megmaradjon, ha a form érvénytelen.

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

## Az old direktíva használata az űrlapban

Amikor a formot elküldjük, de az érvényesítés hibát talál, a create.blade.php űrlapban a value="{{ old('title') }}" és {{ old('text') }} kifejezések megőrzik a felhasználó által beírt értékeket. Ez fontos, hogy ne veszítsük el a már beírt adatokat, ha az űrlap újratöltődik egy hiba miatt.

A következő (bold) kiemelt parancsokkal egészítsd ki a create.blade.php fájlt:

<form action="{{ route('notes.store') }}" method="POST" class="space-y-4">

**@csrf**

<!-- Cím input mező -->

<input

type="text"

name="title"

placeholder="Cím"

class="w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"

**value="{{ old('title') }}"**

**@error('title')**

**<div class="text-red-600 text-sm mt-2">{{ $message }}</div>**

**@enderror**

<!-- Szöveg textarea mező -->

<textarea

name="text"

rows="10"

placeholder="Kezdj el írni ide..."

class="w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent resize-none"

**>{{ old('text') }}</**textarea>

**@error('text')**

**<div class="text-red-600 text-sm mt-2">{{ $message }}</div>**

**@enderror**

<!-- Mentés gomb -->

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

Mentés

</x-primary-button>

</form>

## Hibakezelés a Kontrollerben

A kontrollerben biztosítjuk, hogy a validációs szabályok megfelelően be legyenek állítva.

Térjünk vissza a NoteController.php store metódusához: használjuk az Eloquent Buildert és a save() metódust:

public function store(Request $request)

{

$request->validate([

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

'text' => 'required|string',

]);

**Note::create([**

**'title' => $request->title,**

**'text' => $request->text,**

**'user\_id' => auth()->id(),**

**]);**

**return redirect()->route('notes.index')->with('success', 'Bejegyzés sikeresen hozzáadva.');**

**//** *dd($request); ne felejtsd el kikommentezni*

}

1. Ha üresen küldjük el az űrlapot, akkor a validáció hibát generál, és a mezők alatt piros színnel jelennek meg a hibaüzenetek.
2. Ha kitöltjük a title mezőt, de a textarea mezőt üresen hagyjuk, akkor a cím megmarad, és csak a textarea alatt jelenik meg a hibaüzenet.
3. Sikeres mentés esetén a felhasználó az üzenetek oldalra kerül átirányításra, ahol a sikeres mentésről kap értesítést.

Ezzel a megoldással a Laravel lehetőségeit kihasználva, a komponensek és a hibakezelés egyszerűen és átláthatóan kezelhetők, megőrizve a már beírt adatokat, és biztosítva, hogy a felhasználó könnyen lássa a hibákat.

# Tömeges Hozzárendelés Védelme az Eloquent Modellekben

Laravelben a modellek alapértelmezés szerint védve vannak a tömeges hozzárendeléstől, hogy elkerüljük a nem kívánt vagy rosszindulatú adatok mentését. Ahhoz, hogy egy modell engedélyezze a tömeges hozzárendelést bizonyos mezőkre, két lehetőség van:

1. **$fillable**: Ezzel a védett változóval megadhatjuk, hogy mely mezők lehetnek kitölthetők tömeges hozzárendeléssel.
2. **$guarded**: Ezzel megadhatjuk, hogy mely mezők ne legyenek kitölthetők tömeges hozzárendeléssel (ha üresre állítjuk, akkor minden mező kitölthetővé válik).

**Teendő a Note.php Modell Fájlban**

Nyisd meg az app\Models\Note.php fájlt, és illeszd be a következő kódot a tömeges hozzárendelés engedélyezéséhez.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Note extends Model

{

use HasFactory;

// Engedélyezett mezők tömeges hozzárendeléshez

protected $fillable = ['title', 'text', 'user\_id'];

}

* **$fillable**: Ezzel csak a title, text, és user\_id mezők válnak elérhetővé tömeges hozzárendeléssel, ami biztonságosabb, mert kizárja az összes többi mezőt.
* **$guarded = []**: Alternatív megoldásként használhatjuk ezt az üres guarded változót is, ami azt jelenti, hogy minden mező kitölthető. Ez azonban csak akkor ajánlott, ha biztosak vagyunk benne, hogy minden mező biztonságosan kezelhető tömeges hozzárendelés esetén.

Ez a beállítás megakadályozza, hogy az alkalmazás tömeges hozzárendelési sebezhetőségnek legyen kitéve, így csak a megengedett mezőkön keresztül történhet adatbevitel.

**Tesztelés**

1. Mentsd el a módosításokat a Note.php fájlban.
2. Frissítsd az alkalmazást a böngészőben, és próbálj meg egy új bejegyzést hozzáadni az űrlapon keresztül.
3. Ha minden jól működik, a bejegyzés elmentődik anélkül, hogy más, nem engedélyezett mezők módosulnának.

Ezután frissítsünk a böngészőben és kattintsunk valamelyik bejegyzés címére.

1. Egyedi azonosítók

Adjunk a jegyzeteinkhez egyedi azonosítót, a könnyebb lekérdezés érdekében.

Távolítsuk el, az összes fejegyzést az adatbázisból.

A terminálon írjuk be:

php artisan migrate:rollback

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

Amikor lefuttatunk egy migrációt (például php artisan migrate), a Laravel létrehoz egy új bejegyzést a migrations táblában. Ez a bejegyzés nyomon követi, hogy mely migrációk futottak le és milyen sorrendben. A migrate:rollback parancs ezt a nyilvántartást használja a legutóbb lefuttatott migrációk azonosítására és azok visszavonására.

Ha például így adjuk meg: php artisan migrate:rollback --step=2 a –step meghatározza, hogy hány lépést (batch-t) vonjon vissza.

Alapértelmezés szerint ez a parancs az utolsó migrációs "batch"-et vonja vissza, ami lehet egy vagy több migráció is, ha azok egyszerre lettek futtatva.

**Kapcsoldó parancsok:**

* **php artisan migrate**: Futtatja az összes függőben lévő migrációt.
* **php artisan migrate:reset**: Visszavonja az összes migrációt.
* **php artisan migrate:refresh**: Visszavonja az összes migrációt, majd újra futtatja őket.
* **php artisan migrate:fresh**: Törli az összes táblát az adatbázisból, majd újra futtatja a migrációkat.

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. Store function:

use Illuminate\Support\Str;

Note::create([

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

A böngészőben, hogy ne az id jelenjen meg, ezért nyissuk meg az **index.blade.php**-t és javítsuk az id-t uuid-ra: (a címben is és a tovább gombnál is javítani kell)

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

Ezért a **NoteController.php** fájl show metódusát is javítani kell:

public function show($uuid)

{

// A note rekord keresése a uuid alapján és a user\_id ellenőrzése

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

return view('notes.show', ['note' => $note]);

}

A web-php-ben is szükséges az útvonalat javítani:

//Üzenet részletezése uuid()-vel

Route::get('/notes/{uuid}', [NoteController::class, 'show'])->where('uuid', '[0-9a-fA-F\-]+');

A model-ben a **Note.php** –hez hozzá kell adni, hogy automatikusan generálja le a uuid-it:

use Illuminate\Support\Str;

// Automatikusan generál UUID-t az 'uuid' mezőhöz, ha új rekord jön létre

protected static function boot()

{

parent::boot();

static::creating(function ($model) {

if (empty($model->uuid)) {

$model->uuid = (string) Str::uuid();

}

});

}

**boot metódus**: Ezzel a metódussal a modell automatikusan lefuttat egy creating eseményt minden új rekord létrehozásakor.

**Str::uuid():** A Laravel Str segédosztályával generálhatunk egy új UUID-t, amit az uuid mezőbe mentünk el, ha az üres.

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 a uuid alapján, de ezt a lépést kihagyhatjuk, ha még elviselünk némi módosítást a **NoteControllerben**:

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);**

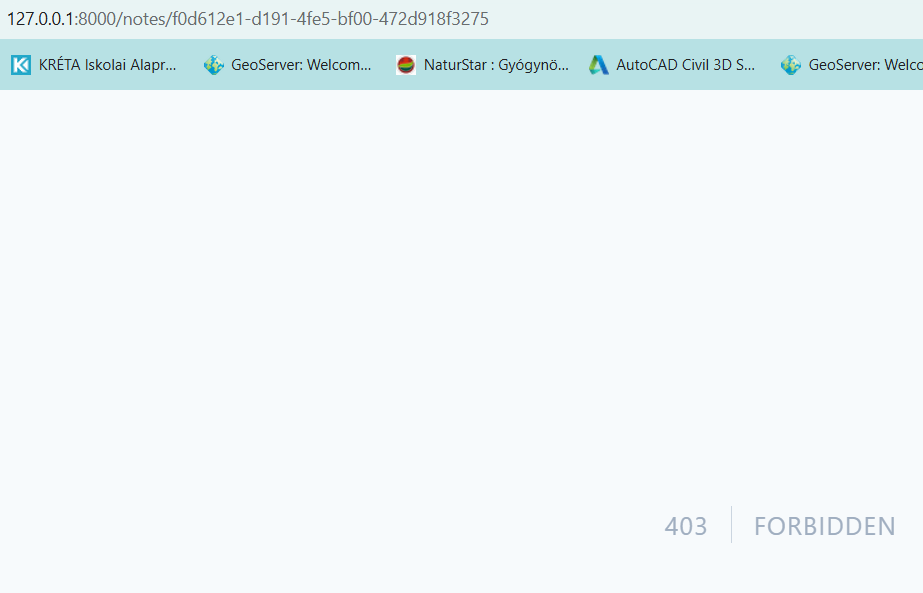
**}**

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.

**Tesztelés:**

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.

# Bejegyzések szerkesztése

Mostmár, a felhasználónk 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.

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:

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.

<x-app-layout>

<x-slot name="header">

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

**{{ \_\_('Szerkesztés') }}**

</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">

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

**@method('put')**

@csrf

<!-- Cím input mező -->

<input

type="text"

name="title"

placeholder="Cím"

class="w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"

**value="{{ old('title', $note->title) }}"/>**

@error('title')

<div class="text-red-600 text-sm mt-2">{{ $message }}</div>

@enderror

<!-- Szöveg textarea mező -->

<textarea

name="text"

rows="10"

placeholder="Kezdj el írni ide..."

class="w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent resize-none"

**>{{ old('text', $note->text) }}</textarea>**

@error('text')

<div class="text-red-600 text-sm mt-2">{{ $message }}</div>

@enderror

<!-- Mentés gomb -->

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

Mentés

</x-primary-button>

</form>

</div>

</div>

</div>

</x-app-layout>

A NoteController.php update metódusát egészítsük ki:

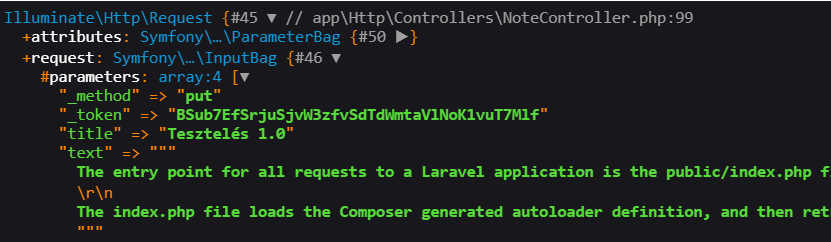
public function update(Request $request, $id)

{

dd($request);

}

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

 Ha a request-ben megtaláltuk a bejegyzést, törölhetjük vagy kommentezzük a dd()-t.

Először is szükségünk lesz az authorizációra, hogy a felhasználónk belépett és jogosan használja az oldalt, a NoteControllerben 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.

# Bejegyzések törlése

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>

**action attribútum:** A törlés útvonalat (notes.destroy) határozza meg, amelyhez a route segédfüggvényt használjuk, és átadjuk a törlendő $note modellt.

**@method('delete'):** Mivel az űrlapok alapértelmezett metódusa POST, ezzel a direktívával jelezzük a Laravelnek, hogy a kérést DELETE-ként kezelje.

Ahhoz, hogy a két gomb egymás mellé kerüljön, alkalmazhatjuk a Tailwind flex osztályát:

<div class="flex items-center space-x-4">

+ záró tag.

**flex:** Ez a div elem belsejében az összes elemet egy sorba rendezi.

**items-center:** Függőlegesen középre igazítja az elemeket a div belsejében.

**space-x-4:** A space-x-4 osztály hozzáad egy kis távolságot (például 1rem) a gyermekelemek között, így a Szerkesztés és Törlés gombok között is lesz távolság.

A css-hez is tudunk hozzáadni némi stílust, hogy a gombok konzisztensebben jelenjenek meg:

.btn-link {

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

}

.btn {

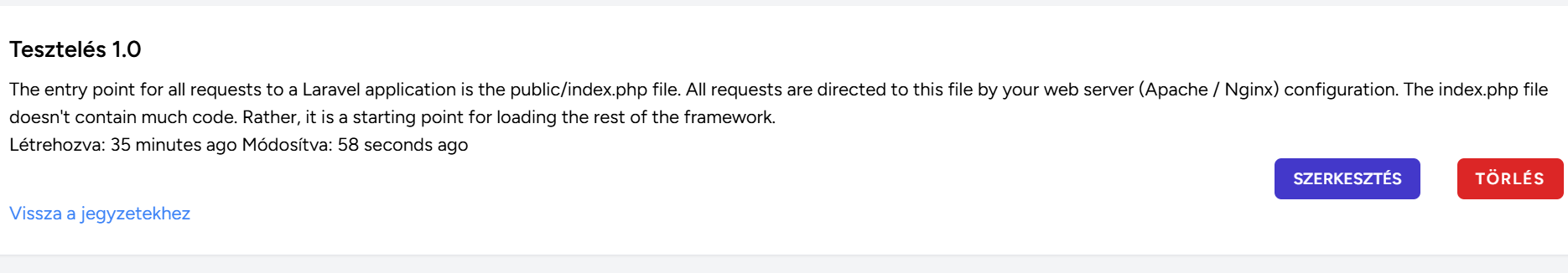
@apply px-4 py-2 rounded-md font-bold text-sm text-white uppercase tracking-widest focus:ring-2;

}

.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>

A confirm() egy felugró ablakban megjeleníti az üzentet.

A resources/views/layouts/**app.blade.php** fájlod <head> szakaszába helyezd el a SweetAlert2 linket:

<!-- SweetAlert2 CSS és JS CDN linkek -->

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/sweetalert2@11/dist/sweetalert2.min.css">

<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

Törlés Gomb Módosítása a SweetAlert2-vel:

onclick="confirmDelete()

Az app.blade.php-nek a fejlécébe megadhatod a scriptet:

<script>

function confirmDelete() {

Swal.fire({

title: 'Biztosan törölni szeretné?',

text: "Ez a művelet nem visszavonható!",

icon: 'warning',

showCancelButton: true,

confirmButtonColor: '#d33',

cancelButtonColor: '#3085d6',

confirmButtonText: 'Igen, törlés',

cancelButtonText: 'Mégse'

}).then((result) => {

if (result.isConfirmed) {

document.getElementById('deleteForm').submit();

}

});

}

</script>

**confirmDelete() függvény**: A SweetAlert ablakot a confirmDelete függvény hívja meg, és csak akkor indítja el az űrlap elküldését, ha a felhasználó a confirm (OK) gombra kattint.

**document.getElementById('deleteForm').submit();:** Ez az utasítás csak akkor fut le, ha a result.isConfirmed igaz értékkel tér vissza, vagyis ha a felhasználó ténylegesen megerősítette a törlést.

Így viszont, a gombot is módosítanunk kell, ugyanis ez azért történik, mert a form elküldése automatikusan történik, mielőtt a SweetAlert megerősítést kapná. A megoldás az, hogy a törlés gombot egy külön eseménykezelőben hívjuk meg, és a SweetAlert válaszától függően indítjuk el a form elküldését.

<!-- Törlés gomb -->

<button type="button" class="btn btn-danger ml-4" onclick="confirmDelete()">Törlés</button>

<!-- Törlés űrlap -->

<form id="deleteForm" action="{{ route('notes.destroy', $note) }}" method="post" style="display:none;">

@method('delete')

@csrf

</form>

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);

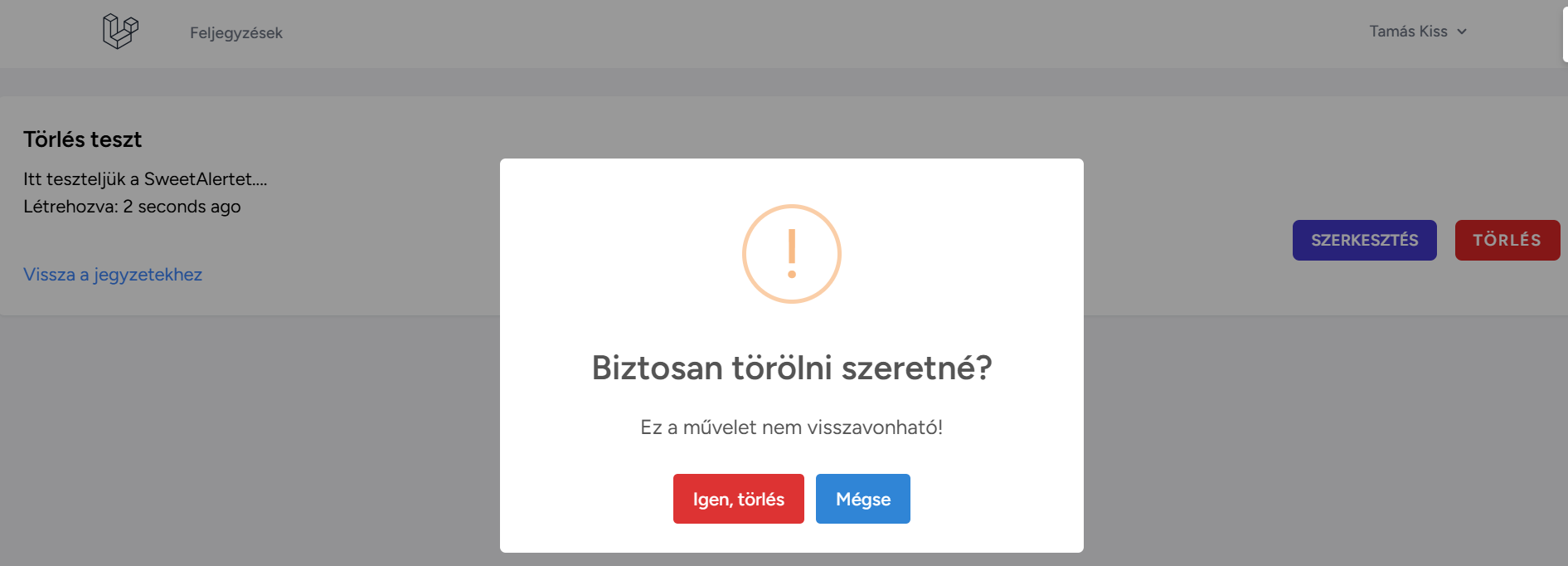
}

$note->delete();

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

}

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



# Visszajelzések a usernek a végrehajtott műveletek sikerességéről

## Értesítés a módosítás 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">

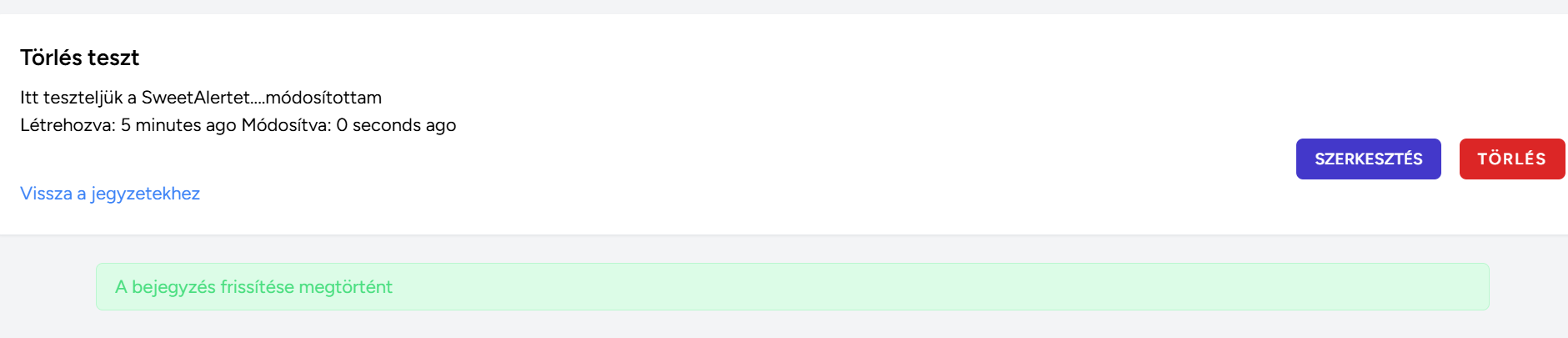
@if(session('success'))

<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.



Az új, újrafelhasználható alert-success komponens létrehozása lehetővé teszi, hogy könnyen jelenítsd meg a sikerüzeneteket az alkalmazás különböző részein anélkül, hogy ismételned kellene a HTML kódot.

A resources/views/components/ mappában hozz létre egy új fájlt alert-success.blade.php néven, és helyezd el benne a következő kódot:

@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

**@if(session('success')):** Ellenőrzi, hogy van-e success kulccsal egy üzenet a session-ben, és csak akkor jeleníti meg a div tartalmát, ha igen.

**$slot:** Ez egy beépített változó a Laravel komponensekben, amely lehetővé teszi, hogy a komponens használatakor tartalmat adjunk meg, ami automatikusan belekerül ebbe a helyre. Ebben az esetben a session('success') üzenet jelenik meg itt.

Most, hogy elkészült a komponens, használhatod bármelyik nézetben. A korábbi kód helyére ezt írd be:

<x-alert-success>

{{ session('success') }}

</x-alert-success>

Ez a kód megjeleníti a success üzenetet a session-ból a alert-success komponens formátumával. A komponens automatikusan megjeleníti a kódot a zöld háttérrel és határokkal, ahogyan a alert-success.blade.php fájlban meg van határozva.

Ez a megoldás több okból is jobb, mint az előző:

* Újrafelhasználhatóság: A alert-success komponenst bárhol használhatod az alkalmazásban, ahol sikerüzenetre van szükséged, így elkerülheted a kód ismétlését.
* Könnyű karbantartás: Ha a későbbiekben meg szeretnéd változtatni az üzenet stílusát, elég a alert-success.blade.php fájlt módosítani, és mindenhol frissülni fog.
* Tiszta és olvasható kód: Az <x-alert-success> szintaxis tisztább és könnyebben érthető, mintha minden egyes alkalommal újraírnád a teljes HTML struktúrát.
* Javítja a Laravel alkalmazás moduláris szerkezetét és könnyen kezelhetőbbé teszi a jövőbeni fejlesztéseket is.

## Értesítés a törlés sikerességéről

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:

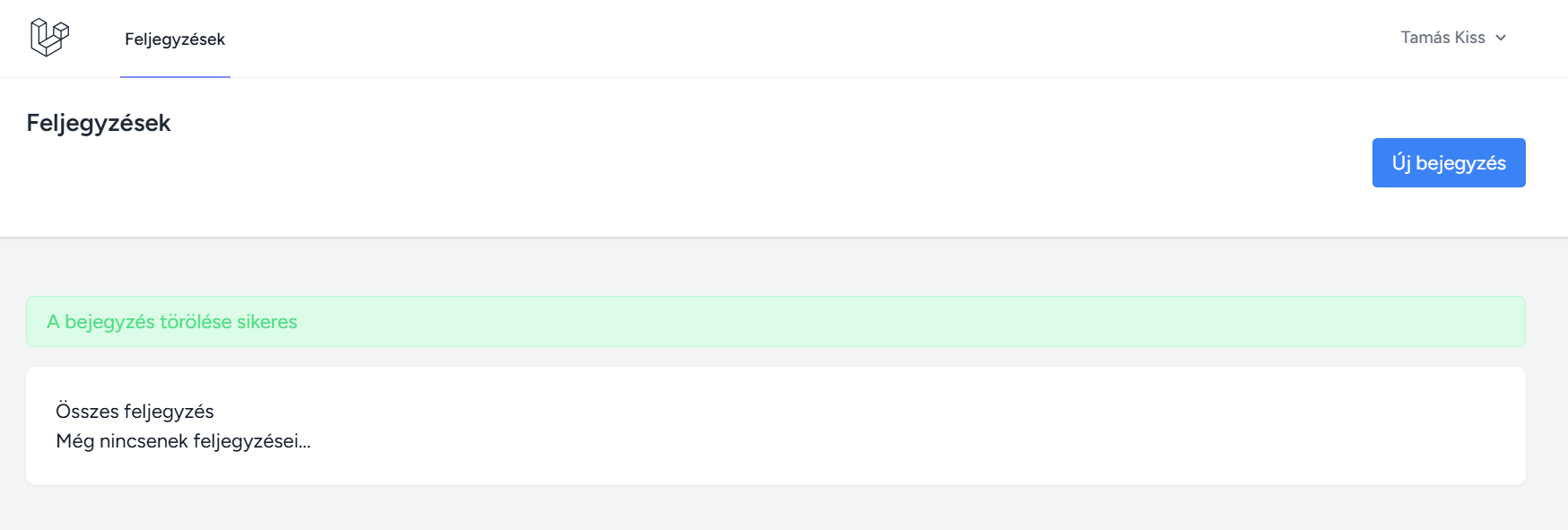
<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

<x-alert-success>

{{ session('success') }}

</x-alert-success>

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



Ezzel el is készült az első alkalmazásunk. ☺

# 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.

Az Eloquent kapcsolatok Laravelben lehetővé teszik, hogy az adatbázistáblák közötti kapcsolatokat egyszerűen, olvashatóan és hatékonyan kezeljük. Az Eloquent különböző típusú kapcsolatokat támogat, amelyek lehetővé teszik a táblák közötti összetett relációk kezelését. Nézzük meg ezeket részletesen:

## Egy az Egyhez (One-to-One)

**Példa:** Egy felhasználónak egy profilja van.

Ez a kapcsolat két tábla között áll fenn, ahol az egyik rekord közvetlenül egy másik rekordhoz kapcsolódik.

Implementáció: Az User és Profile modellek között például így hozhatunk létre egy egy-az-egyhez kapcsolatot:

**User Modell:**

public function profile(){

return $this->hasOne(Profile::class);

}

**Profile Modell:**

public function user(){

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

}

Ezután az User modellből a profile metóduson keresztül érhetjük el a kapcsolódó profilt:

$user->profile.

## Egy a Sokhoz (One-to-Many)

**Példa:** Egy felhasználónak több bejegyzése van.

Ebben az esetben egy tábla egy rekordja több rekordhoz kapcsolódik egy másik táblában.

Implementáció: Az User és Post modellek között például így hozhatunk létre egy egy-a-sokhoz kapcsolatot:

**User Modell:**

public function posts(){

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

}

**Post Modell:**

public function user(){

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

}

Ezután az User modellből a posts metóduson keresztül érhetjük el a kapcsolódó bejegyzéseket: $user->posts.

## Sok a Sokhoz (Many-to-Many)

**Példa:** Egy diák sok tárgyat felvehet, és egy tárgyat több diák is felvehet.

Ebben az esetben két tábla között sok-a-sokhoz kapcsolat van, és egy köztes (pivot) táblára van szükség a kapcsolat megvalósításához.

Implementáció: Az Student és Subject modellek között például így hozhatunk létre egy sok-a-sokhoz kapcsolatot:

**Student Modell:**

public function subjects(){

return $this->belongsToMany(Subject::class);

}

**Subject Modell:**

public function students(){

return $this->belongsToMany(Student::class);

}

Laravelben a pivot táblát automatikusan nevezi el, például student\_subject, de ezt testre is szabhatod, ha különleges elnevezést szeretnél használni.

## Egy a Sok Közül (Has Many Through)

**Példa:** Egy ország sok felhasználóval rendelkezik, és ezeknek a felhasználóknak sok bejegyzése van.

Az egy-a-sok-közül kapcsolat lehetővé teszi, hogy egy modell más modelleken keresztül hozzáférjen egy táblához.

Implementáció: Például, ha van egy Country, User, és Post modell, és egy ország összes bejegyzését szeretnénk lekérdezni:

**Country Modell:**

public function posts(){

return $this->hasManyThrough(Post::class, User::class);

}

**Miért Hasznosak Az Eloquent Kapcsolatok?**

* **Egyszerűbb lekérdezések**: Az Eloquent kapcsolatok segítségével egyszerűen és olvashatóan dolgozhatsz az adatbázissal anélkül, hogy komplex SQL lekérdezéseket kellene írnod.
* **Automatikus adatkötés**: Laravel automatikusan elvégzi az adatkapcsolatokat, így könnyebben hozzáférhetsz a kapcsolódó adatokhoz.
* **Több lehetőség kapcsolatok kezelésére**: Eloquent több kapcsolat típust támogat, így bármilyen adatstruktúrát könnyen lekezelhetsz a modellek között.

Az Eloquent kapcsolatokkal gyorsabban és könnyebben kezelheted az adatbázis táblák közötti kapcsolatokat, mindezt Laravel saját metódusaival. Minden kapcsolati típus (egy-az-egyhez, egy-a-sokhoz, sok-a-sokhoz stb.) lehetővé teszi, hogy egyszerűen navigálj az összetett adatbázisokban anélkül, hogy bonyolult SQL-t kellene írnod.