**Felhasználókezelés**

Szeretnénk tudni felhasználókat regisztrálni, bejelentkeztetni. Ezt a funkcionalitást külső függőségként el tudjuk érni a Laravelen keresztül. (A laravel alá valaki ezt már megírta, majd megosztotta és mi ezt fel tudjuk használni.)

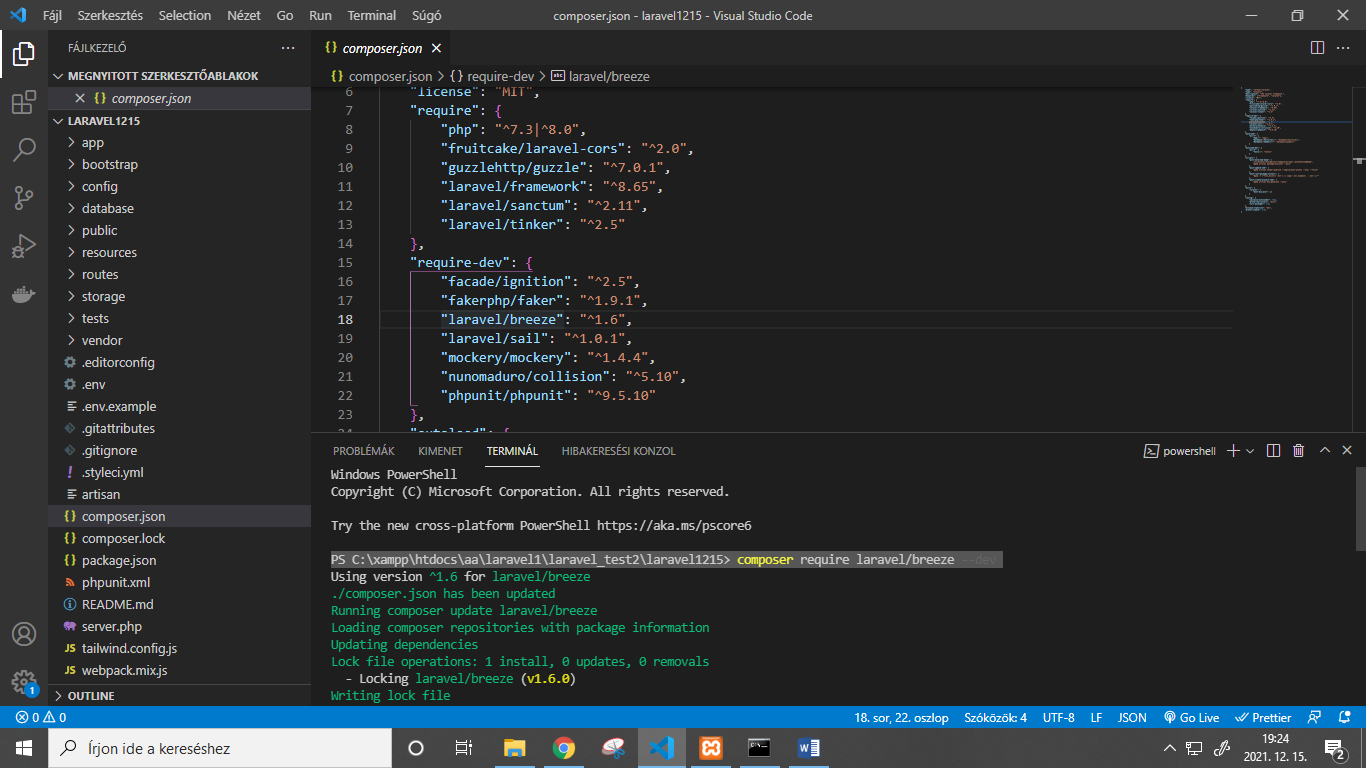
A telepítéshez megtalálható a dokumentáció az alábbi linken. Itt minden információ megtalálható a telepítéshez. Mi a breeze-hez tartozó megoldást választottuk.

Laravel Breeze 🡪 <https://laravel.com/docs/8.x/starter-kits>

Kreálunk egy új Laravel projectet:

composer create-project laravel/laravel laravel1215

Nem kell composer install, mivel az már lefut a projekt kreálásakor



1. lépés: PHP oldalon telepítjük a függőségeket:

PS C:\xampp\htdocs\aa\laravel1\laravel\_test2\laravel1215> composer require laravel/breeze --dev

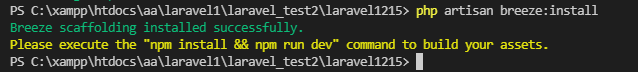
1. lépés: Telepítjük a Breeze-t a projekten belül

PS C:\xampp\htdocs\aa\laravel1\laravel\_test2\laravel1215> php artisan breeze:install

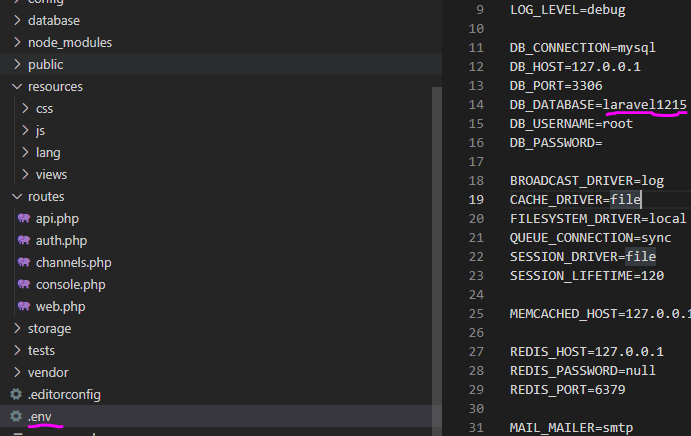
Breeze scaffolding installed successfully.

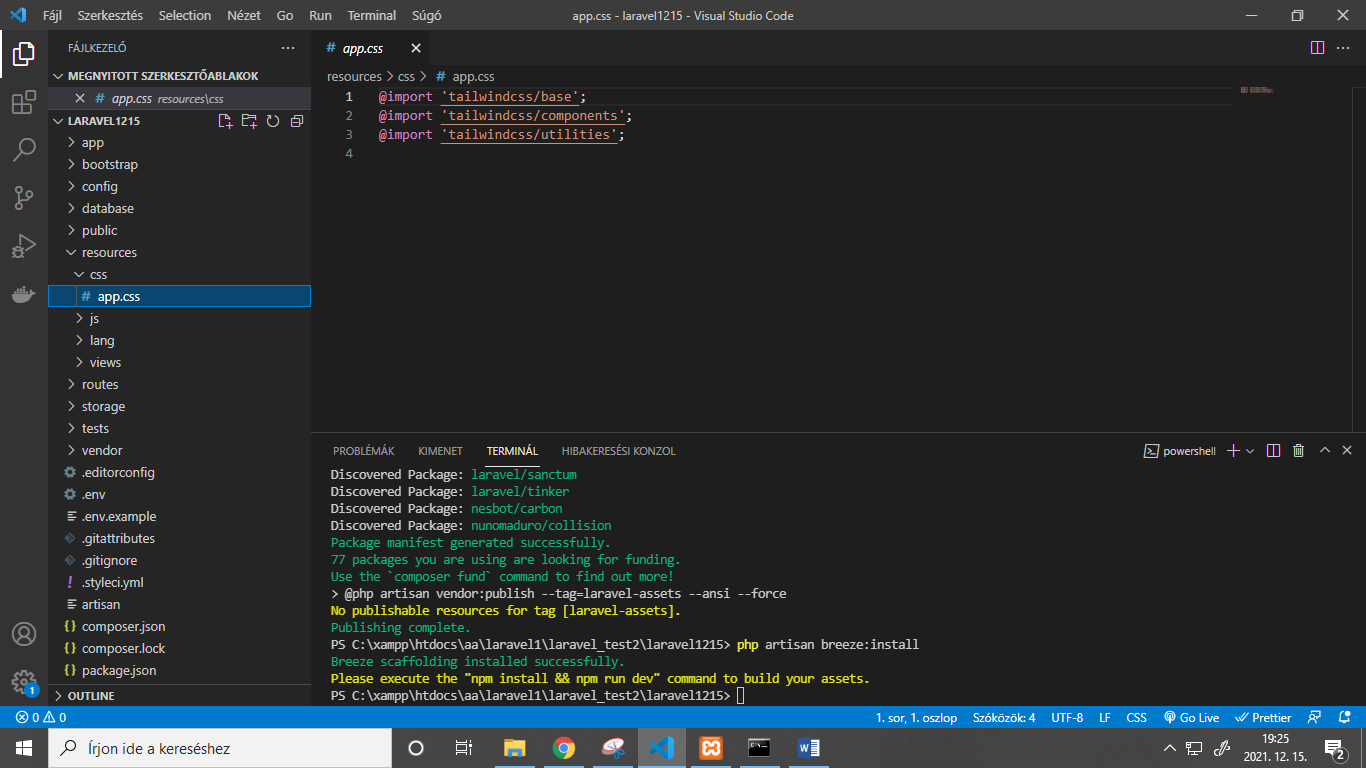
Please execute the "npm install && npm run dev" command to build your assets.

PS C:\xampp\htdocs\aa\laravel1\laravel\_test2\laravel1215>



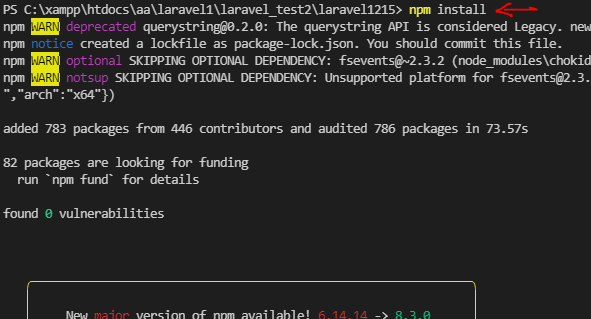
DB kapcsolat előkészítése a projektben (az adatbázisunk neve is *laravel1215* lesz):





1. lépés: Telepítjük a felülethez tartozó függőségeket (Ehhez szükséges egy node megléte, ezen belül is a npm-re lesz szükségünk)

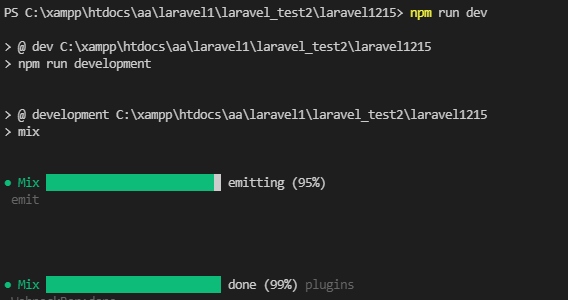
npm install



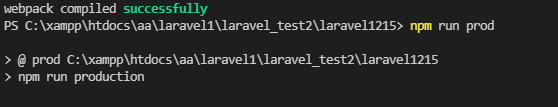
1. lépés: "Lefordítjuk" a komplexebb javascript illetve css fájlokat, a böngésző által is érthető formára

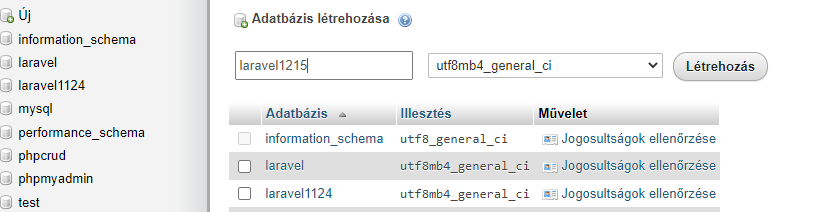
Fejlesztési fordítás: a resources alatti fájlok fordítása

npm run dev 🡪 development, lehetővé teszi a debuggolást



npm run prod 🡪 production, anonimitás

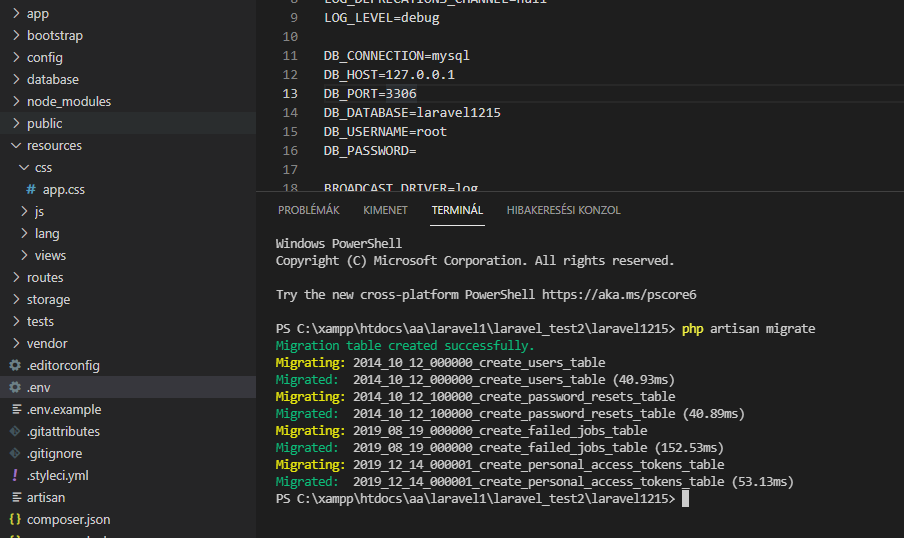




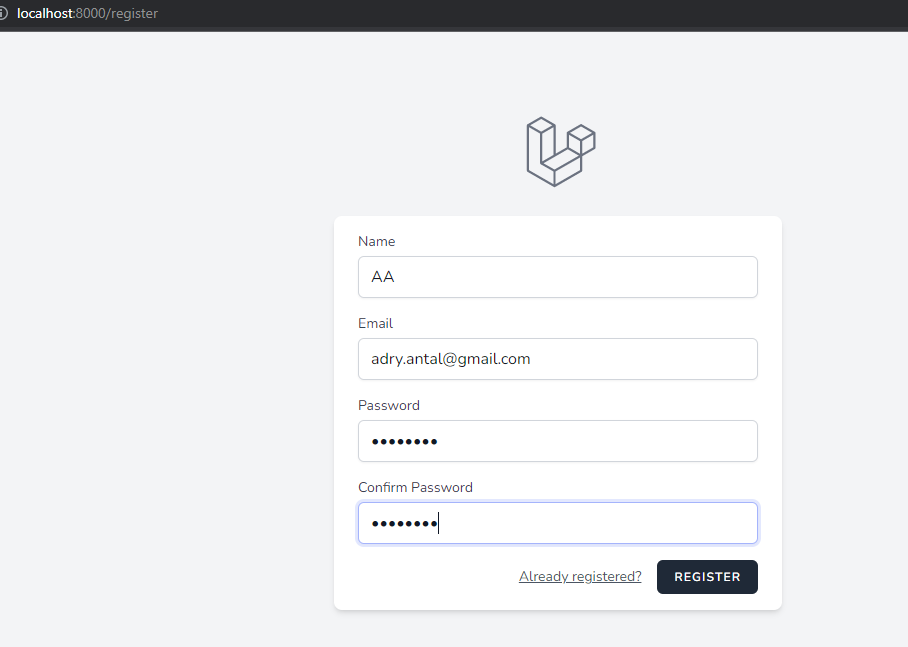
lépés: Végrehajtjuk a felhasználókezeléshez szükséges adatbázis migrációt

php artisan migrate (ez ráér akkor, amikor már megvan az összes táblaattribútum)

php artisan serve



Új regisztráció a <http://localhost:8000/> alatt (<http://localhost:8000/register>):



Amennyiben már megtörtént a migráció (php artisan:migrate), akkor a regisztráció egy új rekordot hoz létre az adatbázisban.

HF:

saját szakdolgozat 🡪 mi történik, amikor több tábla kapcsolódik

egy olyan projekt, amilyet órán csináltunk

lehessen API-ból műveleteket létrehozni, adatokat kezelni

pl. blogbejegyzés, projektek

adatbázis kapcsolatok vizsgálata

**Middleware-ek:**

<https://laravel.com/docs/8.x/middleware>

A middleware-ek közvetlenül a Controllerbe érkezés előtt futnak le és tudnak vizsgálni mindenféle feltételt. Az egyetlen feladatuk azt eldönteni, hogy a felhasználótól érkező kérés megfelel-e a támasztott kritériumainknak. Pl.: Aki a kérést küldte az be van-e jelentkezve a rendszerünkbe? (Nem akarjuk, hogy az adatainkhoz idegen hozzáférhessenek. Vagy authentikáció nélkül nem is tudunk a felhasználó felé adatokat közvetíteni. Pl.: messenger üzenetek (először tudnunk kell, hogy kiről van szó és azután tudjuk visszaadni neki a saját üzeneteit), e-mailek). Másik tipikus eset, hogy ha vizsgálni akarjuk, hogy valakinek van-e admin jogosultsága. (Pl.: egy mezei banki ügyfélnek csak a a saját adatait szabad megtekinteni, viszont egy ügyfélszolgálati munkatársnak rá kell tudnia látni bárkinek az adataira)

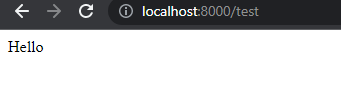
//az útvonal csak akkor érhetó el, ha a middleware-n átmentünk

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

    return 'Hello';

})->middleware('auth');

Csak akkor enged be, ha be vagyunk jelentkezve:



Ha nem, akkor viszadob a login oldalra.

Ha bejelentkezéssel megyek rá az útvonalra, beenged.

Groupolás:

Route::middleware([EnsureTokenIsValid::class])->group(function () {

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

//

});

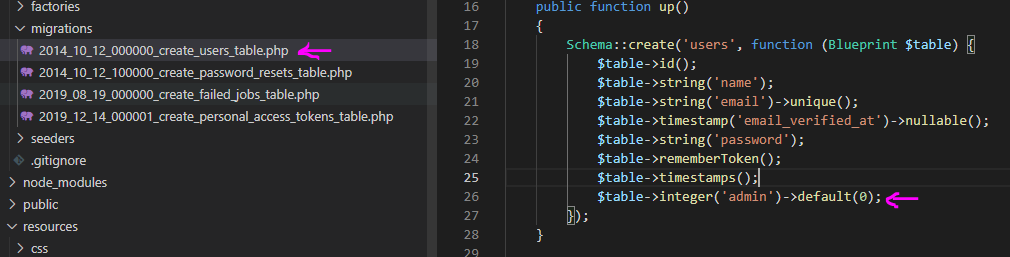
**Feladat**

Szeretnénk admin útvonalakat levédeni az olyan felhasználók elől, akik nem rendelkeznek admin hozzáféréssel.

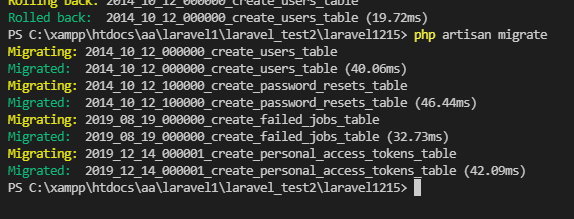
**1. lépés:**

Szeretnénk vizsgálni, hogy a felhasználó admin-e vagy sem. Ezért ezt el kell tárolnunk a rendszerben. Ehhez módosítanunk kell a felhasználókhoz tartozó tábla attribútumait. A **Users** táblához tartozó migrációs fájlt ki kell egészíteni egy plusz *admin* attribútummal. (**database/migrations/2014\_10\_12\_000000\_create\_users\_table.php**)

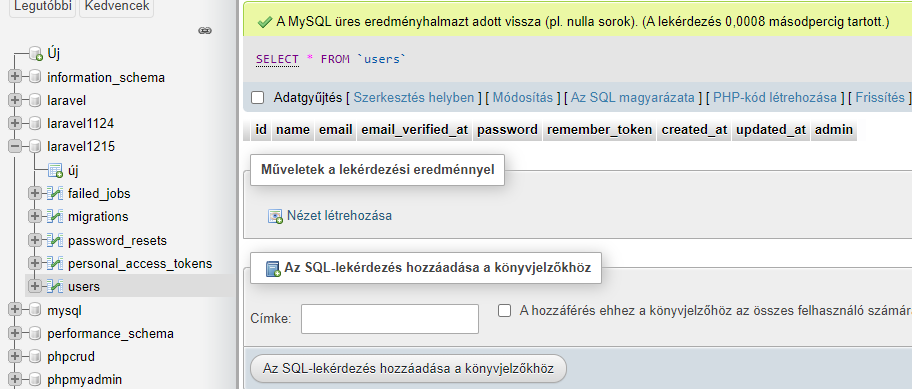
A migrációs fájlba beszúrunk egy extra *admin* attribútumot, mely alatt tároljuk egy adott userről, hogy admin-e vagy nem. Ha igen, akkor az értéke 1, ellenkező esetben pedig 0 lesz. A default értéket 0-ra álllítuk be.



Jöhet a migráció: php artisan migrate



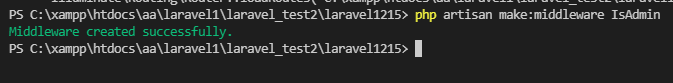
Ha módosul valami az adatbázisban, és rollback + új migration kell, akkor új regisztráció(k)ra is szükség lesz, hiszen az addig tárolt adatok törlődnek!

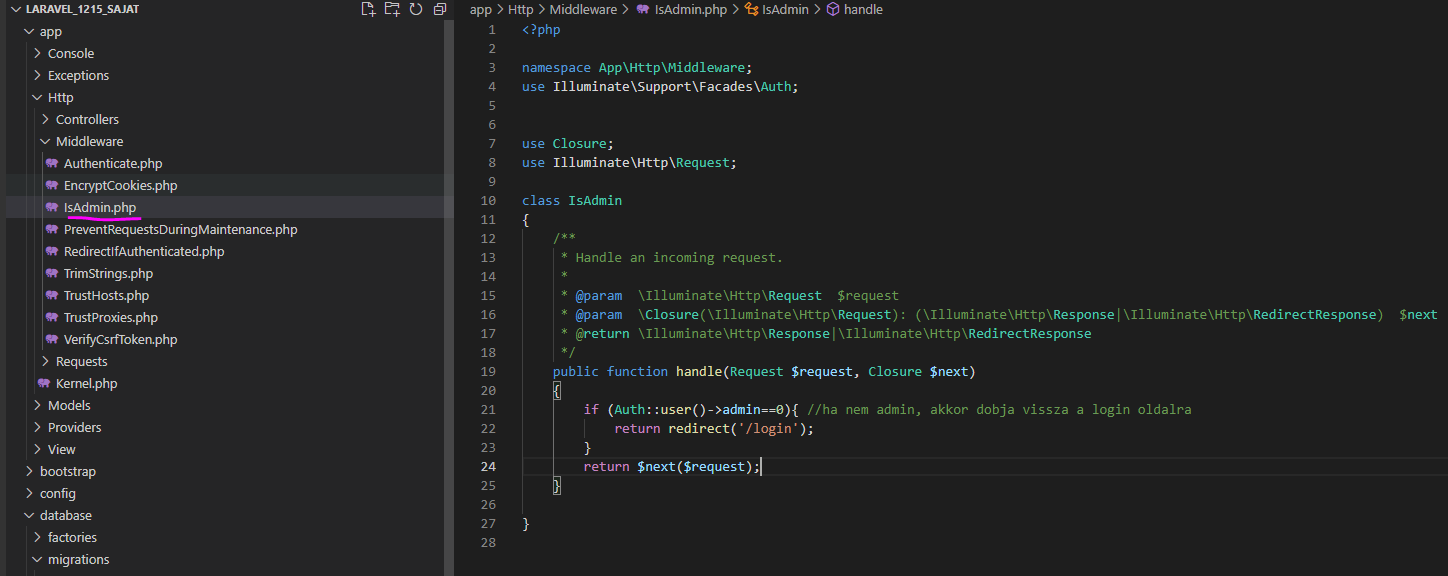


### 2. lépés:

Létrehozzuk a logikát tartalmazó middleware osztályt. Ez a fájl (IsAdmin.php) az **app\Http\Middleware** mappa alatt jön létre:

PS C:\xampp\htdocs\aa\laravel1\laravel\_test2\laravel1215> php artisan make:middleware IsAdmin





### 3. lépés:

A middleware-ünknek a logikája mindig a handle függvényben kerül implementálásra. Ez a függvény mindig két paramétert fog kapni bemeneti paraméternek. Az egyik a *request*, amit a felhasználó küld a szerver felé (pl.: **GET /api/projects**), a másik pedig egy függvény lesz. Amikor azt szeretnénk, hogy a middleware engedje tovább a kérésünket a controller felé, akkor az alábbi értékkel kell visszatérnünk: return $next($request);

Amikor legeneráljuk az új middleware-ünket, akkor gyakorlatilag ennyi lesz a handle függvényén belül, ami átenged minden kérést mindentől függetlenül.A mi üzleti logikánk szerint ez a middleware csak akkor fogja továbbengedni a felhasználókat, ha ők admin jogosultsággal rendelkeznek, egyébként irányítsuk át egy másik felületre.

Az authentikációból kikérjük, hogy a user admin-e. Ha nem, irányítsa vissza a login oldalra.

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

    {

        if (Auth::user()->admin==0){ //ha nem admin, akkor dobja vissza a login oldalra

            return redirect('/login');

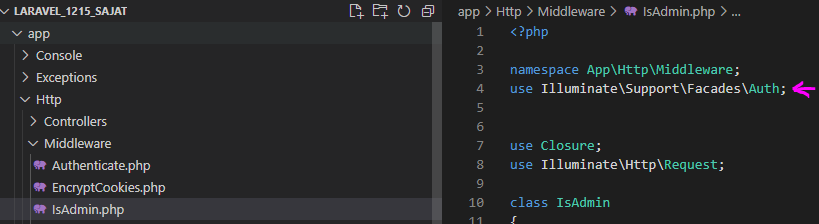
        }

        return $next($request);

    }

**Részletezve:** felhasználjuk a laravel által alapértelmezetten kínált **Auth** osztályt, amely vizsgálja a kérést végző személyt. Az **Auth** oszálynak van egy statikus függvénye (azt **::** operátorral tudjuk meghívni), ami **user()** névre hallgat. Ezzel ki tudjuk kérni a felhasználóhoz tartozó adatbázis rekordot. Innentől tudjuk vizsgálni az adatbázisban tárolt attribútumait. Az első lépésben létrehoztunk egy **admin** attribútumot, amit jelen helyzetben fel is tudunk használni. Vizsgáljuk a felhasználó ezen attribútumának az értékét, amennyiben ez **hamis (vagyis a felhasználó nem admin)**, akkor őt átirányítjuk a kezdőoldalra, egyébként meg továbbadjuk az adatokat a következő lépésnek. **Összefoglalva:** a handle logikájából kijön az, hogy az egyetlen kritérium, amit ennek az eszköznek vizsgálnia kell az az, hogy a felhasználó admin attribútumának mi az értéke. Ha ez számunkra elfogadható értékkel rendelkezik, akkor tovább engedjük a felhasználót, ha pedig nem, akkor átirányítjuk (ebben az esetben) vagy hibát dobhatunk, vagy éppenséggel egy 404-es oldalra dobhatjuk a felhasználót ( **abort(404);** ) Ahhoz, hogy a fenti logika működjön még tudatnunk kell a middleware osztályunkkal, hogy a felhasznált **Auth** osztályt hol tudja megtalálni, ehhez a következő sort kell beszúrni a middleware fájl fejlécébe:

use Illuminate\Support\Facades\Auth;



**4. lépés**

Az egyes útvonalakhoz hozzá kell még rendeljük a middleware-t, amelyek esetében vizsgálni akarjuk a megadott feltételt. Ezt a routes/web.php alatt kell hozzárendelni az egyes útvonalakhoz. Erre két lehetőségünk van.

1. Egy útvonalhoz hozzárendeljük a middleware-ünket.

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

    return 'Hello';

})->middleware(['auth', IsAdmin::class]);

1. Egy csoportot hozunk létre, amelyen belül minden útvonalra vonatkozni fog a middleware feltétele.

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

    Route::get('/test2', function() {

        return 'Hello 2';

    });

    Route::get('/test3', function() {

        return 'Hello 3';

    });

});

A fenti esetben mind a /test2 mind a /test3 útvonalakra vizsgálni fogjuk az admin jogosultságot.

**Kiegészítés**

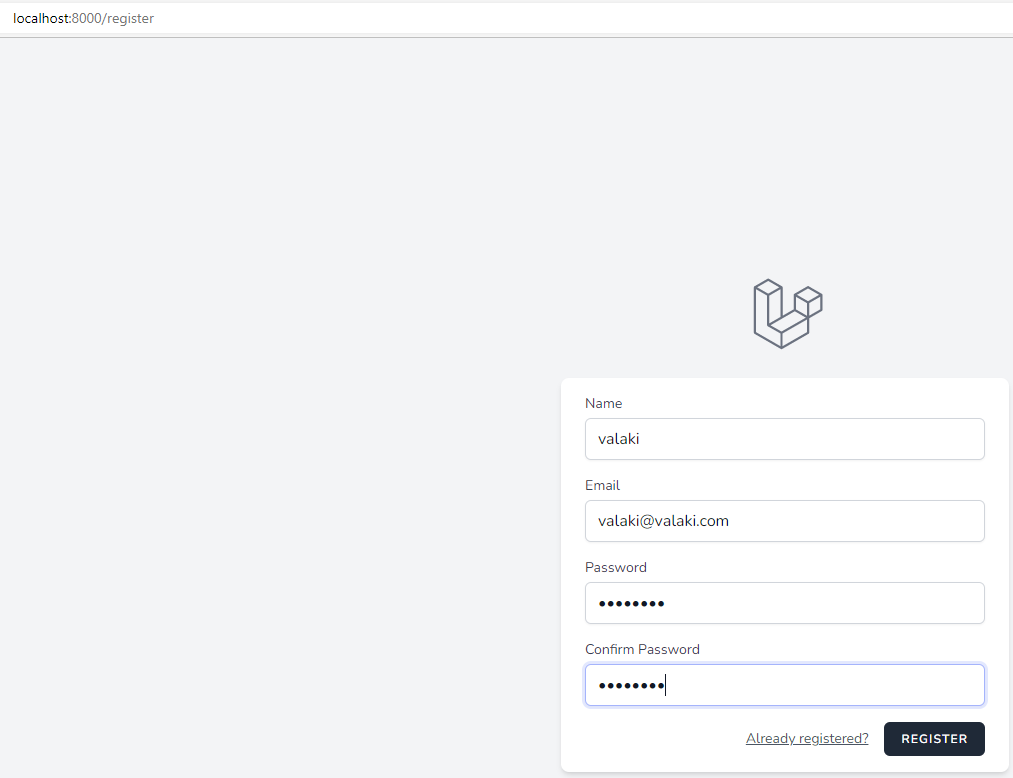
Saját middleware-t a fent leírt lépésekkel tudunk létrehozni, azonban a laravel kínál nekünk előre megírt middleware-eket. A legáltalánosabban használt middleware az **'auth'** middleware. Ez az eszköz egyszerűen azt vizsgálja, hogy a felhasználó be van-e jelentkezve vagy sem. Általánosságban érdemes ezt a middleware-t alkalmazni az összes olyan útvonalon, ahol az **Auth** osztályt használjuk, mivel ha az **'auth'** middleware-en átmegy a kérés, akkor az **Auth** osztályt is meg fogjuk tudni hívni hiba nélkül. Tipikus ilyen eset lehet, amikor egy **/api/projects** útvonalon nem a létező összes projektet akarjuk kilistázni, hanem a bejelentkezett felhasználóhoz tartozó összes projektet.

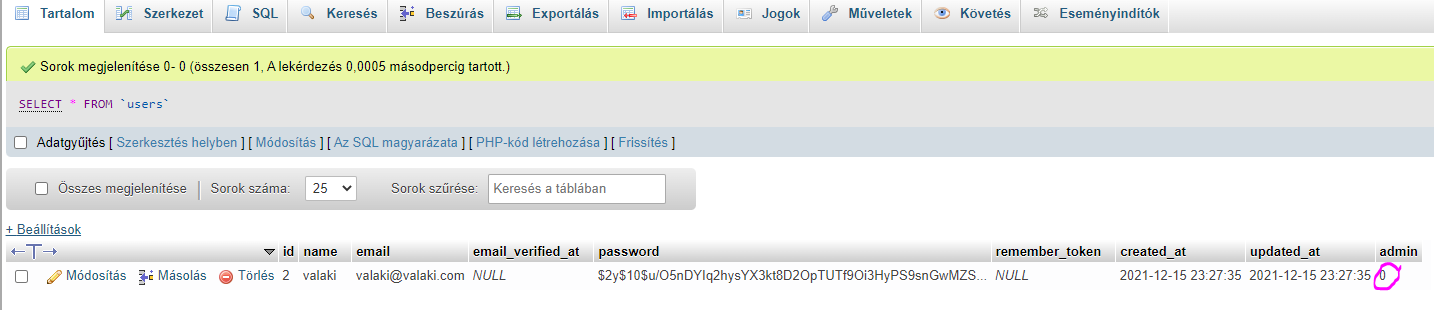
A példaprojektben az admin jogosultságot csak adatbázis táblában lehet módosítani, így tesztelés céljából ott kell átírni az adott user admin jogát. Ez alapján lehet tesztelni a mintakódot.

**Tesztelés felhasználókkal; adminokkal és nem adminokkal egyaránt**

<http://localhost:8000/register> linken keresztül regisztrálhatunk tesztadatokat. Mint korábban említettük, minden regisztráció új rekordot hoz létre az adatbázis ’users’ táblájában. (Az *admin* attribútum értékét – tesztelésképp – akár manuálisan is változtatgathatjuk a <https://localhost/phpmyadmin> -on belül.)

Regisztrálok egy új usert, aki NE legyen admin:





Teszteset1: nincs belogolva a user

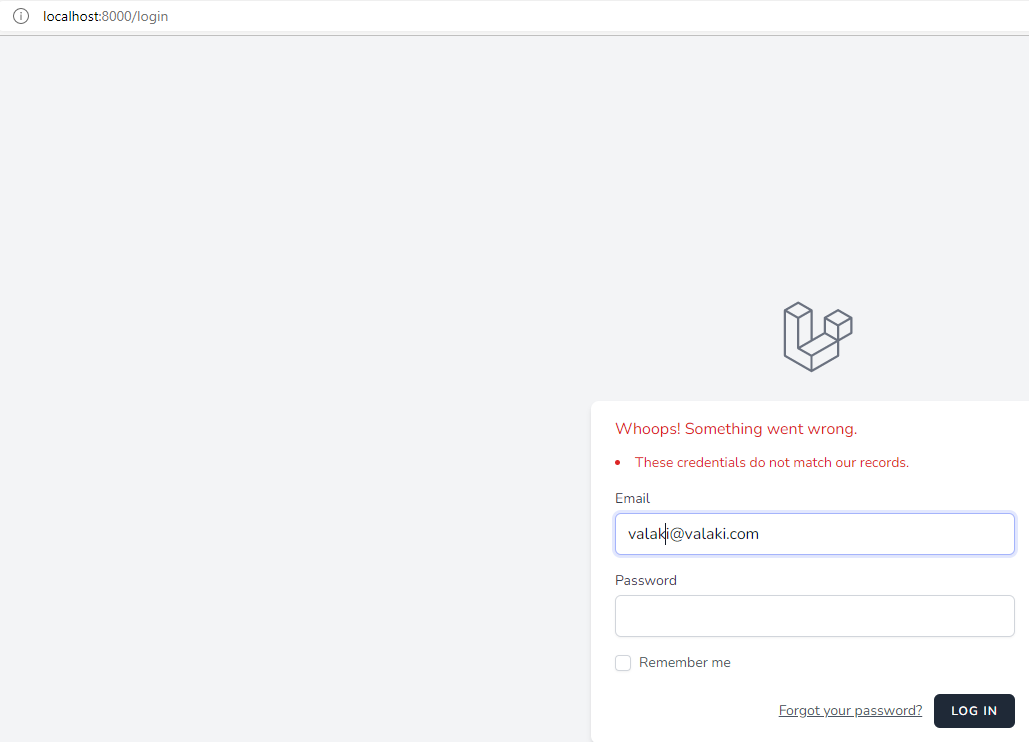
Teszt útvonal megnyitása:

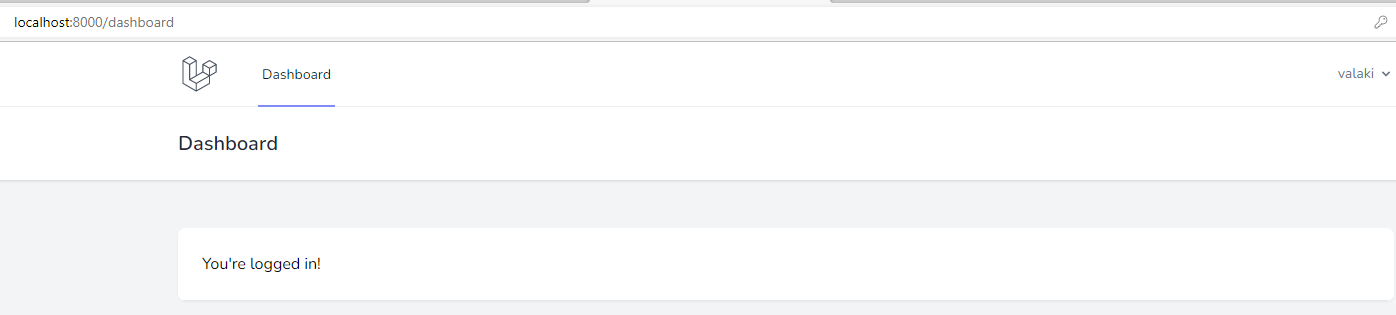


Vissza lett irányítva a login oldalra:



Teszteset2: be van logolva a user, aki nem admin

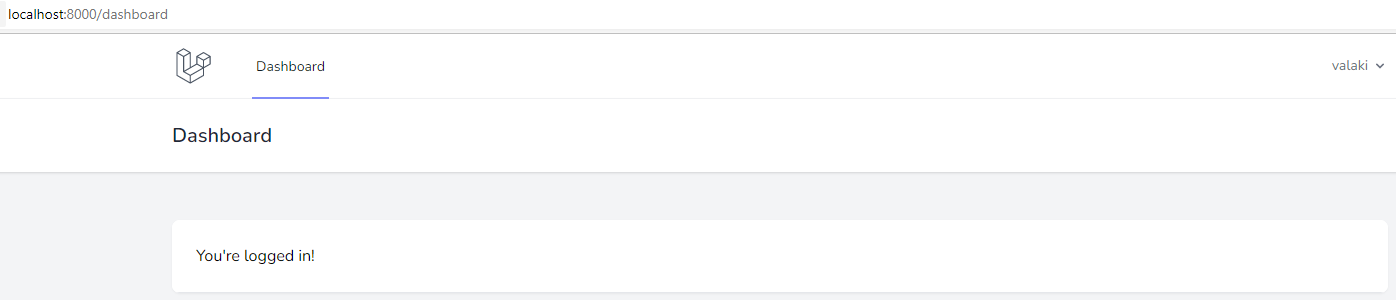




Teszt útvonal megnyitása:



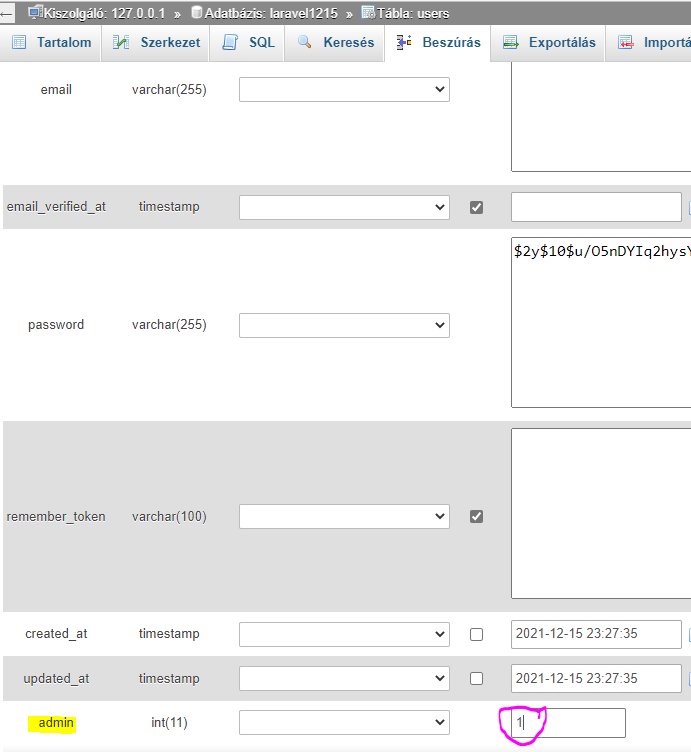
Ismét visszadobja, viszont mivel már be van jelentkezve, nem a login oldalra, hanem az „üdvözlőoldalra” küldi:

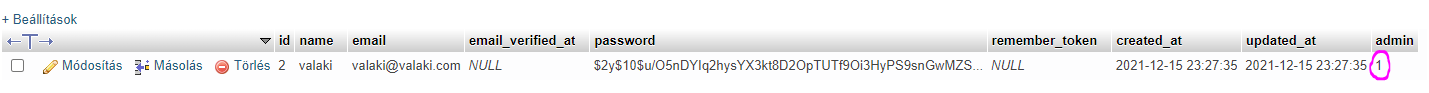


Tehát nem engedi át.

Teszteset2: be van logolva a user, aki admin

Átírom az admin attribútum értékét a tesztrekordban:





Teszt útvonal megnyitása:



Átengedte a felhasználót:

