## Alapfeltételek:

### PHP + Adatbázis (XAMPP csomag tartalmazza, DB: MySQL)

### composer - php-s külső függőségek menedzseléséhez szükséges kis program.

Telepítés: <https://getcomposer.org/download/>

A composer parancs minden mappában elérhető parancs lesz.

Futtatásakor mindig az aktuális parancssori könyvtárban fog dolgozni. composer install parancs esetén egy composer.json fájlt fog keresni.

**Laravel - keretrendszer**

Projekt létrehozás: laravel projektet a composer program segítségével tudunk létrehozni az alábbi parancs futtatásával:

composer create-project laravel/laravel *myproject*

Laravel projekt klónozása meglévő repository-ból:

.env.example fájl tartalmának másolása .env fájlba

.env módosítása, hogy a kapcsolódási paraméterek legyenek élesek (DB kapcsolódási paraméterek beállítása)

composer install

php artisan migrate

Laravel projekt elindítása:

php artisan serve

**Egy adatbázis tábla létrehozása, valamint a hozzá tartozó adatok létrehozása, módosítása, törlése API útvonalakon keresztül**

**Előkészítés:**

php artisan make:model Project -m // -m jelenti, hogy a modellhez migrációt is létre kívánunk hozni

php artisan make:controller ProjectController -m Project // -m Project a modellhez kapcsolódást jelenti

**Fájl elérési helyek:**

migrációs fájlok: database/migrations

model fájlok: app/Models

controller fájlok: app/Http/Controllers

view fájlok: resources/views

Ezekkel előkészülnek a kiinduláshoz szükséges fájlok.

**Lépések:**

1. migration fájl szerkesztése, a megadott attribútumok létrehozása.

2. php artisan migrate

3. ha később módosul a migration tartalma és már migráltunk, akkor php artisan migrate:rollback paranccsal vissza tudjuk vonni a korábbi migrációt és az új struktúrát létre tudjuk hozni a php artisan migrate parancs ismételt futtatásával.

**API útvonalak előállítása**

Az összes elem kilistázása JSON formátumban:

Útvonal: GET - /api/projects

Controllerben:

return response()->json(Project::all()); vagy

return Project::all();

Egy elem kilistázása JSON formátumban:

Úrvonal: GET - /api/project/{id}

Controllerben:

return response()->json(Project::find($id)); vagy

return Project::find($id);

Egy elem létrehozása:

Útvonal: POST - /api/project

Controllerben:

$project = new Project();

$project->attributum = $request->attributum;

$project->save();

return Project::find($project->id);

Adott elem szerkesztése:

Útvonal: PUT - /api/project/{id}

Controllerben:

$project = Project::find($id);

$project->attributum = $request->attributum;

$project->save();

return Project::find($project->id);

Adott elem törlése:

Útvonal: DELETE - /api/project/{id}

Controllerben:

$project = Project::find($id);

return $project->delete();

A html-t tartalmazó útvonalak definiálása:

Létrehozó form:

Útvonal: GET - /project/new

Controllerben:

return view(project.new); -> a view függvényen belül gyakorlatilag a mappasturktúra képződik le, ebben a példában a views mappa alatt a projekt mappában lévő new.blade.php ra hivatkozunk.

Módosító form:

Útvonal: GET - /project/edit/{id}

Controllerben:

return view(project.edit);

Összes adat listázása oldal:

Útvonal: GET - /projects

Controllerben:

return view(project.list);

**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.)

**Dokumentáció**

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

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

**Telepítés**

A felhasználókezelést érdemes a projekt létrehozása után közvetlenül megcsinálni (kvázi projekt létrehozása utáni első lépés). Később összeakadhat az általunk írt kódokkal.

1. lépés: PHP oldalon telepítjük a függőségeket

composer require laravel/breeze --dev

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

php artisan breeze:install

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

npm run dev

1. lépés: Végrehajtjuk a felhasználókezeléshez szükséges adatbázis migrációt

php artisan migrate

Ezek után elérhetőnek kell lennie a felhasználókezelés minden funkciójának. Ami legáltalánosabban szükséges, az a bejelentkezés és a regisztrációs funkcionalitás. Természetesen a hattérben megoldódnak a cookie és session kezeléshez tartozó feladatok is.

**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)

**Dokumentáció**

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

**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 attribútummal. (**database/migrations/2014\_10\_12\_000000\_create\_users\_table.php**)

$table->boolean('admin')->default(false);

**2. lépés:**

Létrehozzuk a logikát tartalmazó middleware osztályt. Ez a fájl az **app\Http\Middleware** mappa alatt jön létre

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.

if (Auth::user()->admin == false) {     return redirect('/'); } 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 'Ez egy teszt üzenet'; })->middleware(IsAdmin::class);

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

Route::middleware([IsAdmin::class])->group(function () {     Route::get('/test2', function() { return 'Teszt üzenet 2'; });     Route::get('/test3', function() { return 'Teszt üzenet 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.