# Laravel 9 - Alapok

Tartalom

[Laravel 9 - Alapok 1](#_Toc129884681)

[1. VSC extensions 1](#_Toc129884682)

[2. A Laravel alapjai 1](#_Toc129884683)

[3. Routes 6](#_Toc129884684)

[4. Blade template 7](#_Toc129884685)

[5. Controllers 11](#_Toc129884686)

[6. Laravel konfigurációja 12](#_Toc129884687)

[7. Adatbázis kapcsolása 13](#_Toc129884688)

[8. Migrációk létrehozása és futtatása 16](#_Toc129884689)

[9. Laravel Eloquent – ékesszóló :D laravel 18](#_Toc129884690)

# VSC extensions

Érdemes pár extensiont hozzá adni a VSC-hez; Laravel Blade Snipets (Winnie Lin); Laravel Blade Spacer (Austen Cameron); Laravel Blade (Christian Howe).

Laragon : <https://laragon.org/> (hogy ne tegyünk tönkre a xampp-ban a mappákat)

# A Laravel alapjai

A Laravel egy MVC keretrendszer. Az MVC rövidítés a Model-View-Controller hármast fedi. Röviden-tömören annyit jelent, hogy három alapvető részre bontja az alkalmazásodat. Ezen felül rengeteg kisebb-nagyobb része van a keretrendszernek, ami megkönnyíti a munkádat. A lényegesebbekről:

**Model**

Modelleket használsz az adatok eléréséhez. Laravelben egy nagyon jó absztrakciós réteg, az Eloquent ORM található. Egy-egy modelt egy adatbázis táblához vagy nézethez képzelj el (persze nem kötelező adatbázist használnod). A Model neve megegyezik a tábla nevével, példányai pedig egy-egy adatbázis rekordnak felelnek meg. Minden adatbázis műveletet el tudsz végezni az Eloquent segítségével, ráadásul nagyon egyszerűen, de ezt később látni is fogod. A modeleknek nincs előre létrehozva mappájuk, én az app/Models alá szoktam őket helyezni.

**View**

A nézetek gyakorlatilag magát a HTML kódot tartalmazzák. A Laravel a nézetekhez a Blade sablonrendszert használja. A Blade segítségével tudod a dinamikus adatok kezelését, beszúrását elvégezni, illetve lehetőséget ad különböző vezérlési szerkezetek használatára és más nézetek beszúrására. A nézeteket a resources/views mappában találod. Lényegében az összes html fájl itt található. A blade arra kiváló, hogy a html-t és a php-t kicsit finomabb módon tudjuk együtt használni, mint a natív php esetében. pl ne kelljen minden php kód elé kezdő zárótag, stb.

A resources mappába kerülnek a js-ek, css-ek,

A lang mappába vannak a nyelvesítsére vonatkozó fájlok. Alapértelmezetten angol van benne, ha akarunk magyart, akkor csak egy másolatot kell róla készíteni, és átnevezni hu-ra, és át kell írni benne a szövegeket magyarra. Az app.php fájlunkban, ha a nyelvet magyarra állítjuk, és talál magyar fordítást, akkor az üzeneteket magyarul fogja megjeleníteni.

**Controller**

A nézetek és modellek közötti kapcsolatot tartja fent. Minden lekérés egy controllerbe fut be. A controller tartalmazza a program logikáját: itt töltöd be adatbázisból az adatokat, itt számolsz velük, majd a végén átadod egy nézetnek amit megkap a felhasználó. A kontrollerek az app/Http/Controllers mappában találhatóak.

**Routing**

A Router feladata a beérkező kérések megfelelő irányítása. Ha pl. van a weboldaladon egy /contact oldal, fel kell vedd, hogy melyik Controllerhez irányítsa a Router. A routeok ettől persze állhatnak több dologból: megadhatsz változókat, állíthatsz be Middlewareket, amik szűrik a kéréseket, illetve azt is itt adod meg, hogy melyik route milyen kéréseket fogadjon (POST, GET, stb.). A routeokat a routes mappában található fájlokban tudod szerkeszteni. Egyelőre neked csak a web.php-val kell foglalkoznod.

**Migration**

Az adatbázisok kezelése, míg egyedül dolgozol egy projekten, elég egyértelmű. Ha viszont egy komplex projekten csapatnyi fejlesztő dolgozik, problémás lehet a szerkezeti módosítások követése, egymás között frissen tartása. Az adatbázis migrációk a Laravel query builderére építve tartalmazzák az adatbázisok tábláinak, kapcsolatok beállításainak minden egyes lépését, amik oda-vissza futtathatóak, azaz visszaállíthatóak lépésenként, minden egyes műveletet úgy visszaállítva, mintha meg sem történtek volna. Ha minden egyes változtatáshoz új migrációt veszel fel, a társadnak csak annyi dolga van, hogy frissíti az adatbázist a php artisan migrate paranccsal. E mellett azért is nagyon hasznos a migrációk használata, mert így szintén ennek a parancsnak a kiadásával létre tudod hozni a teljes adatbázist az alkalmazásod telepítésekor. Ezen a migrációk a database/migrations/ mappába kerülnek.

Tehát migration-okat használunk ahhoz, hogy az adatbázisod sémáját, kapcsolatokat, kezdő rekordokat felvedd. Új migrationt az artisan segítségével tudsz létrehozni, a projekt gyökérkönyvtárában a következő parancs kiadásával:

***php artisan make:migration create\_names\_table***

Az elnevezést mindig ilyen módon válaszd, tehát létrehozásnál create\_, a tábla neve az adatokra utaló és többes számban van, majd az, amit létre akarsz hozni, esetünkben table. Ha módosítani akarsz a szerkezeten, **create\_ helyett válaszd az alter\_ kifejezést** vagy pl. új mező beszúrásához az add\_mezo\_to\_tabla\_table formát.

Ha kiadtad a parancsot, létrejött a database/migrations/ alatt egy új migration a kért névvel.

A fájlon belül két function jön létre:

**up() function**

Ez a migration futásakor fog lezajlani, ide kell megírnod azt, amit maga a migration csinál: táblák létrehozását, mezők létrehozását, stb. Tudsz puskázni az alapból létrehozott táblákból, pl a user táblából.

Az Artisan okos és ha a fenti névkonvencióval hoztad létre a migrationt, egyből készített is neked egy megfelelő keretet tábla létrehozásra: a nevét is megadta, adott hozzá egy AUTO\_INCREMENT-es elsődleges kulcsot és két speciális mezőt a timestamps segítségével (created\_at és updated\_at). Ezek a létrehozási és módosítási időt tartalmazzák, amiket az Eloquent automatikusan kitölt az adott rekord létrehozásánál és szerkesztésénél.

**down() function**

Ez akkor fut le, ha szeretnéd visszaállítani az adott migrationt, azaz visszaállni az adatbázisban egy régebbi verzióra. Tábla létrehozás esetén ez egyértelműen DROP-ot jelent. Ha pl. csak egy új mezőt hozol létre az up()-ban, csak ennek a mezőnek az eltávolítását kell megírni. Ez egyébként nem kötelező, de javallott, hisz így oda-vissza tudsz lépegetni az adatbázisod sémájának állapotai között.

**Schema**

A létrejött kódban egyből láthatod, hogy létrejött a Laravel Schema Builderje, amivel function-ekkel tudsz táblákat összeállítani. Ezt azért alkották így, mert a Laravel nem csak MySQL adatbáziskezelőt támogat, hanem pl. MSSQL-t is, ahol vannak különbségek típusokban, stringekben, stb. Ha a Schema Buildert használod, ezeket a különbségeket a Laravel áthidalja és ugyan úgy fog működni a többi támogatott adatbázis kezelőn az alkalmazásod, mint azzal, amivel éppen írod.

A Schema osztály - nevéből következtethetően - az adatbázis sémák kezelésére készült. Első paramétere a tábla neve, másodikban pedig egy ún. Blueprintet állítunk össze, ami a mezőket és azok paramétereit tartalmazza. A típusokhoz egy-egy ilyen function tartozik, mint az id() vagy timestamps(), bár ezek speciálisak.

Ha pl. egy szöveges mezőt szeretnél létrehozni, folytasd a sorokat így:

***$table->string('mezoneve');***

A különböző paramétereket vagy a functionbe kell megadd vagy találsz hozzájuk helper függvényt. Pl. ha nullázható mezőt akarsz:

***$table->string('mezoneve')->nullable();***

Ha meg akarod szabni a stringed hosszát (VARCHAR(X) MySQL-ben) és akarsz egy alapértelmezett értéket adni:

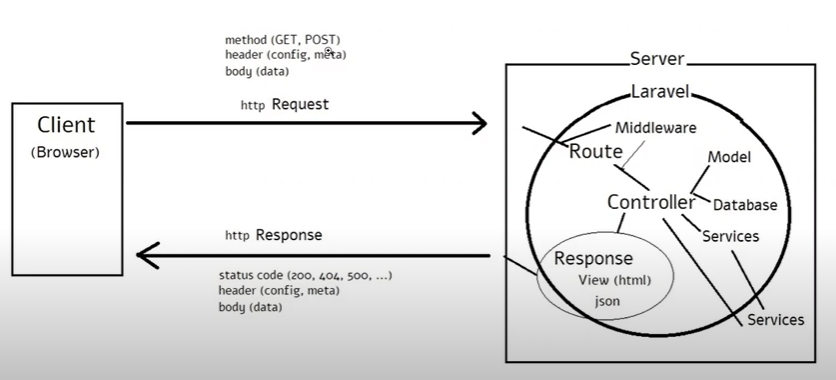
***$table->string('masikmezo', 100)->default('Alapertek');***

**Public mappa**

Kizárólag ezt az egy mappát látja a webszerver. Minden statikus dolgodat (CSS, JS, képek, ikonok, stb.) a public mappába helyezd el. Itt található a Laravel index fájlja is, tehát ha a későbbiekben éles környezetbe akarod helyezni az alkalmazásod, ez kell legyen a gyökérkönyvtára a webszervernek. Érdemes mindent különböző mappákba helyezni ezen belül (/css, /js, /images, vagy ehhez hasonlóak). Ha hivatkozni akarsz pl. egy képre a /public/images/kep.jpg útvonalon, így kell tenned:<img src="/images/kep.jpg">

**Kommunikáció a kliens és a szerver között**

Alapvetően úgy épül fel van egy kliens gép a számítógéped, ahonnan kérést küldesz a szerver felé a böngészőn keresztül, és ha lekérünk valamilyen url címet, akkor az küldeni fog egy http requestet a szerver felé, azt a domain név szerver határozza meg, hogy a beírt cím és a szerver az össze tartozzon. A request 3 dolgot tartalmaz mindig, ez a kérés lehet Get, vagy Post, attól függően, hogy küldünk adatot, vagy várunk adatot. A header infók tartalmaznak mindenféle beállítást, meta adatot, hogy milyen adatot várunk, milyen típusú kérés ez, milyen formátumban fog történni a kommunikáció, stb. A CORS – a cross domain requesteket jelenti, amik elég sok kalamajkát tudnak okozni, el kell küldeni header-ben azt a host-ot amire szeretnéd a választ, és az megnézi klappol e a szerverébe. A body tartalmaz mindenféle adatot, amit esetleg küldünk a request során. Get kérés esetén nincs body, az adat az url-ben utazik. Amikor beérkezik a szerverre a request, ezt első körben a webszerver fogadja, és tovább irányít az ott beállított helyre. pl ha a szerverre befut egy kérés, akkor az melyik könyvtárba irányítódjon, ami viszont nekünk érdekes, amikor a request elérkezik, a laravel applikációnkhoz. Elsőként a route az, ami meghatározza, milyen url-t kért a kliens, de mielőtt elérkeznénk a route-hoz lesznek middleware-k, amelyek ellenőrzik, hogy van e engedélye a kérésnek az adott urlt lekérni, vagy tovább irányítanak máshová. A routera érkezés előtt és után is beékelhetünk különböző middleware-ket. A következő a kontroller, ez végez mindenféle adatfeldolgozást, adatlekérdezést hív meg hozzá, itt találhatóak az adatbázis műveletek, és a kontroller fogja nekünk előállítani a responst. Ez is tartalmaz header-t, body-t ebben lesz a konkrét megjeleníthető weboldal, status, pl ha valami rossz akkor 404 jelenik meg. Lényeg, hogy előállítsa választ, ami lehet egy view egy weboldal, vagy lehet json ha egy api-t írtunk.



**Artisan**

Az artisan a laravel parancskiadó felülete, ha a terminálra beírod, php artisan kiadja az összes lehetséges parancsot, amiből választhatunk. Ha kiadjuk azt a parancsot, hogy php artisan key:generate akkor a .env fájlba be is kerül egy kulcs.

# Routes

Mint az összes keretrendszernek, a Laravelnek is vannak routjai, itt találjuk:

Routes: web.php – itt láthatjuk az alapbeállítást, ami a welcome nézetre vezet, ez egy default route, ami rossz példa, mert a route-ba normál esetben sosem írunk functiont, mert ezt a route-ot a webszerverünk el casheli, és ez nem fog tudni funcitont cash-elni, csak szöveget, úgyhogy nem így fogjuk ezt csinálni, a routokat a controllerek fogják kiszolgálni, de vizsgálódásra egyenlőre megfelel.

Másoljuk le és kicsit módosítsunk:

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

return "<h1>Szia Uram!<h1>";

});

itt a böngészőbe már a crud.test/Home –ot kell beírni, hogy megjelenjen a szöveg.

Ez nem túl praktikus megoldás, ezért használjuk a view-t. A views-t a resources mappán belül találjuk meg. A welcome.blade.php fájlon belül találjuk a welcome page tartalmát.

Amit a 28-ik sor után látunk @ és {} ez a blade szintaxis. A Blade a Laravel által biztosított sablonozó motor. Más népszerű PHP sablonozó motorokkal ellentétben a Blade nem korlátozza a sima PHP kód használatát a nézetekben.

Hozzunk létre egy új fájlt : [home.blade.php](http://home.blade.php)

Módosítsuk az oldalt, írjunk bele egy üdvözlő szöveget és adjuk meg hozzá a helyes routot

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

return view('home');

});

a böngészőben a <http://crud.test/Home> hivatkozással tudjuk megnyitni

A routes view-ban megadhatunk változót is második argumentumként:

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

return view('home', ['name' => "Zita"]);

});

Majd ezt követően a [home.blade.php-ben](http://home.blade.php-ben) pedig írassuk ki: <h1>Szia <?php echo $name ?></h1>

A változók nevét megadhatjuk göndör zárójelekben is, ugyanúgy fog működni:

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

# Blade template

A Blade is egy sablonrendszer, ami jelentősen megkönnyíti a programozás folyamatát - azért is jó, mert a statikus tartalmakat dinamikussá varázsolhatod. Ha készítesz egy sablont, feldarabolhatod apróbb részekre: magára a HTML keretre, a menüsávra és más ismétlődő részekre. Csinálhatsz “komponenseket”, amik olyan apró, gyakran használt részei az alkalmazásodnak, amiket egyszerűen beágyazhatsz, amikor csak kell, akár paraméterek átadásával. A templating rendszer része az is, hogy változók értékeit szúrd be egyes helyekre, feltételeket szabj meg vagy akár egy egész tömbön iterálj végig, hogy pl. táblázatot rajzolj ki.

A Blade szintaxis egyszerű, de könnyen kezeli a beágyazott sablon öröklődését és az összetett feltételes feltételeket. És mivel a Blade szintaxisa PHP-kódba van fordítva, majd gyorsítótárazva, jobb webszerver teljesítményt kínál, mint a szabványos PHP önmagában. A blade view fájlok a .blade.php fájlkiterjesztést használják, és általában a resources/views könyvtárban tárolódnak.

**Alapvető Blade kulcsszavak**

Minden kulcsszó a @ jellel kezdődik. Használatuk olyan, mintha functionöket hívnál meg; pl. a nezet2 Blade beszúrásához a @include(‘nezet2’) kulcsszót használhatod. Ha bővíteni akarsz egy másik nézetet (ezt mindjárt elmagyarázom), arra az @extends(‘fonezet’) való. Feltételeket és ciklusokat rendre az @if() - @endif, @for() - @endfor, @foreach() - @endforeach parancsokkal teheted meg.

Alkalmazzuk a blade-t: illeszd be a [bladetest.blade.php-be](http://home.blade.php-be)

@if($name == "Zita")

<h1>Szia Admin</h1>

@else

<h1>Szia User</h1>

@endif

**Feladat:** Írj hozzá megfelelő route-ot, hogy működjön az oldal.

A blade templatettel megjeleníthetünk adatokat, vezérlési struktúrákat, és layoutot készíthetünk.

Általában minden oldalnak vannak olyan részei, mint header, main, footer stb. A blade template alkalmazza az öröklődést.

Próbáljuk ki a tailvindcss-t. <https://tailwindcss.com/> írjuk be a tailwindtest.blade.php-be

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<script src="https://cdn.tailwindcss.com/3.2.4"></script>

</head>

<body>

<nav class="bg-gray-100 px-8 py-4 text-gray-700 flex items-center justify-between">

<span class="font-bold text-2xl">Brand</span>

<span>Szia Uram!</span>

</nav>

<section class="p-12 mx-auto max-w-6xl text-gray-800">

<h1 class="text-4xl font-bold">Home</h1>

<p class="mt-4 text-lg">Lorem ipsum dolor sit amet consectetur adipisicing elit. Cum odit nobis necessitatibus Repellendus exercitationem, accusantium ducimus sunt fugiat itaque neque accusamus magnam voluptas quidem ipsam nihil labore facere debitis praesentium voluptatem eius optio quos corrupti laborum iste!</p>

</section>

</body>

</html>

**Feladat:** add hozzá a route-ot, hogy a végpont neve: <http://crudfirst.test/Tailwind> legyen.

Egész jó kis templatet kaptunk. Viszont, ha többször szükségünk van erre az elrendezésre, nem túl jó ötlet újra és újra bemásolni.

Hozzunk létre egy új mappát a views mappában: layouts néven, és ezen belül hozzunk létre egy base.blade.php új oldalt. Másold át bele a [home.blade.php](http://home.blade.php) oldal tartalmát. Csak a section tartalmát a h1 és p tag tartalmat rakjuk vissza a home-ba.

A működéshez definiálni kell, a section directoryt. Meg kell határoznunk azt is, hogy melyik fájltól fogja örökölni az elrendezést: @extends('layouts.base') jelen esetben a base.blade.php fájlból a tempaltet, amiben a html tagek is találhatóak. Ha további részleteket szeretnénk csatolni azt az @include-al fogjuk megtenni. @yield a dokumentum tartalmát fogja beemelni,a mit a section tag-ek között kell megadnunk. Hozz létre egy about.blade.php-t:

@extends('layouts.base')

@section('content')

@yield('content');

@endsection

Ha most megnézzük a kezdő oldalt, akkor látható, hogy ugyanúgy megjelenik minden.

Következő lépésként, a componets/base fájlban, a navbarban lévő User (Szia Uram!-ot) nevet, helyettesítsük @yield('name') -el.

A home.blade.php oldal második sorába pedig adjuk meg a nevet: @section('name', 'Zita')

Ugyanúgy szépen meg fog jelenni a név a jobb felső sarokban.

Hozzunk létre a views mappán belül egy új fájlt, about.blade.php néven, és másoljuk át a home tartalmát bele (majd ezt át alakítjuk). Ehhez be kell állítanunk a routot is, nyissuk meg a web.php oldalt.

Adjuk hozzá az új útvonalat

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

return view('about');

});

majd nyissuk meg a <http://crud.test/about> oldalt.

Adjunk hozzá egy footert is, hozzunk létre resources\views\layouts\footer.blade.php fájlt, és másoljuk bele:

<footer class="bg-gray-200 text-center lg:text-left" style="position:fixed; bottom:0; width:100%">

<div class="text-gray-700 text-center p-4" style="background-color: rgba(0, 0, 0, 0.2);">

© 202 Copyright:

<a class="text-gray-800" href="https://tailwind-elements.com/">Tailwind Elements</a>

</div>

</footer>

Egészítsük ki az about.blade.php fájlunkat vele:

@extends('layouts.base')

**@section('name', 'Zita')**

@section('content')

@yield('content')

@endsection

**@include('layouts.footer')**

**Blade components**

Hozzunk létre a views mappában egy components nevű új mappát, és másoljuk bele a layouts mappában lévő fájlt a base.blade.php-t. Csak a tartalmat cseréljük ki benne: (töröljük a @yield-et)

<span>Szia {{ $name }}!</span>

{{ $content }}

A [home.blade.php-ben](http://home.blade.php-ben) adjuk hozzá a base componenst, a dokumentum elejére

nyitó tagnek: <x-base> és a végére záró tagnek: </x-base>

A név megadására pedig használjuk a slot taget: <x-slot name="name"> Zita </x-slot>

Végül cseréljük ki a @section-t is slot tagre: <x-slot name="content"> és a tartalom végén is: </x-slot>

Összességében így kerüljön a fájlba a kód:

<x-base>

<x-slot name="name">

Zita

</x-slot>

<x-slot name="content">

<h1 class="text-4xl font-bold">Home</h1>

<p class="mt-4 text-lg">Lorem ipsum dolor sit amet consectetur adipisicing elit. Cum odit nobis necessitatibus Repellendus exercitationem, accusantium ducimus sunt fugiat itaque neque accusamus magnam voluptas quidem ipsam nihil labore facere debitis praesentium voluptatem eius optio quos corrupti laborum iste!</p>

</x-slot>

</x-base>

Ha újra megnézzük a <http://crud.test/Home> oldalt, az elrendezés változatlanul jól jelenik meg.

**Routes**

Nézzük a web.php-t amely az útvonalakat tartalmazza, az eredeti útvonal, a welcome oldalra vezetett. Az útvonalak nagyon logikus módon kerülnek felépítésre, pl megadhatod a „/” jel után hogy login, vagyis egy belépés oldalra vezessen, ugyanakkor tartalmazza azokat a logikai elemeket, amelyek ellenőrzik, hogy helyesek e a megadott adatok a belépéshez.

Kérdések:

Hol melyik mappában és melyik fájlban tudod megadni az útvonalakat?

Hogyan adhatsz meg egyszerűen egy útvonalat egy fájlhoz?

# Controllers

A laravel controllerei az app mappán belül, a http mappán belül a Controllers mappán belül található controllers.php.

Készítsünk egy új vezérlőt a terminálunkból az artisan paranccsal.

A Laravel generátor eszköze az artisan, amivel a legtöbb objektum váza egyszerűen létrehozható

Alkalmazása:

php artisan [utasítás] [--paraméterek]

Például a legegyszerűbb, a Laravel verzió lekérdezése, írd be a terminálra:

php artisan -v

**Objektumok létrehozása:**

php artisan make:[objektum típus] [--paraméterek]

artisan make:controller - -help a helpereket mutatja meg

php artisan make:controller [kontroller\_fájl\_neve]

Adjuk ki a php artisan make:controller WelcomeController parancsot a terminálban. Az app –http – controllers mappán belül meg kell jelennie a WelcomeController.php fájlnak. Igaz ezt kézzel is hozzá adhatjuk, de az artisannal gyorsabb, mert nem kell keresgélned a hol van a controllerek mappája, és ha bele kukkantasz ki is van már egészítve néhány sorral.

A WelcomeControllerbe írjunk egy index nevű függvényt:

public function index(){

return view('welcome');

}

A web.php-ben is adjuk meg az új route-ot, most csak ez maradjon benne: (többit kommentezd ki)

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

ha megnézed a nyitó lapot, még nem jelenik meg a nyitó lap, nagyon fontos, hogy hozzá add az oldalhoz az osztályt!!! Jobb egérgombbal klikk a WelcomeControllerre és ellenőrizd az oldal tetején bekerült e + egy sor:

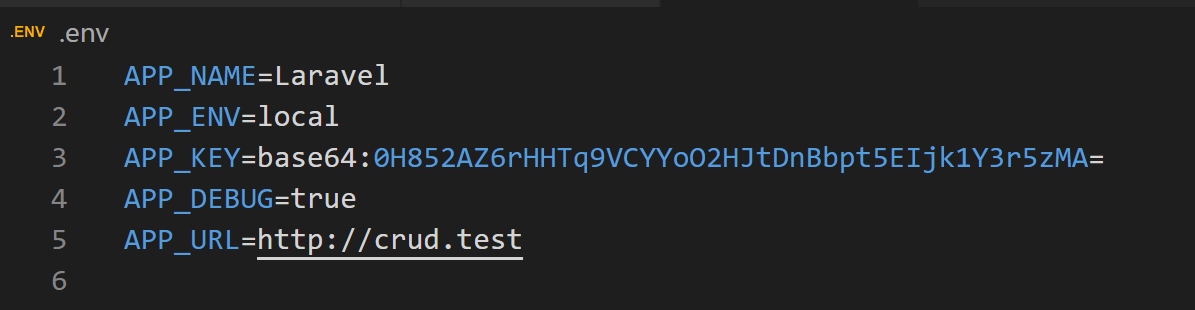
use App\Http\Controllers\WelcomeController;

megjelenik a laravel welcome oldala.

# Laravel konfigurációja

Minden alkalmazásnak van egy csomó konfigurációs beállítási értéke, amelyeket különböző fájlokból kell elérni az alkalmazásban, például az alkalmazás neve, az adatbázis hitelesítő adatai, a levelezési szolgáltatás hitelesítő adatai és így tovább. Ezeknek a környezeti változóknak a kezelésének elősegítése érdekében a Laravel egy .env kiterjesztésű fájlt tartalmaz az alkalmazás gyökerében.

Kattints rá a .env fájra és ott láthatod az alkalmazásokhoz kapcsolódó változókat, például a név, a környezet, az egyes projektekhez egyedi alkalmazáskulcs, az APP\_DEBUG érték, amely helyi környezetben igaz, élesben pedig false. Ez határozza meg, hogy a hibainformációk mekkora része jelenjen meg a felhasználó számára.

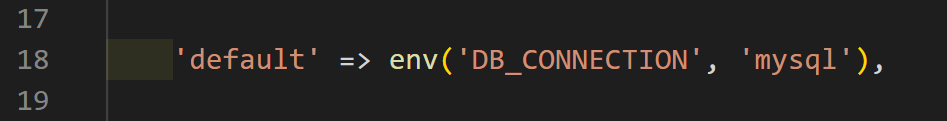


Az APP\_URL az alap URL. Éles környezetben ez tartalmazza a domain nevét. Alatta az adatbázis-kapcsolati értékeket láthatod, beleértve a kapcsolat típusát, felhasználónevet, jelszót és így tovább.

Ha olyan verziókezelést használsz, mint a Git, fontos, hogy ez a fájl nem lesz véglegesítve a tárhelyen. Ez a fájl automatikusan a .gitignore része. Ez biztonsági okokból és azért is van így, mert a csapat minden tagja más konfigurációs értékkel rendelkezhet, és az éles kiszolgáló is, ezért van egy env.example fájlunk, amely az .env fájlhoz hasonlóan tartalmazza az összes változó nevét, ahol meg kell változtatni az értékeket.

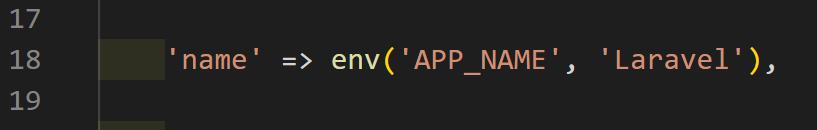
Ez a fájl a tárhoz van kötve, és a csapat minden tagja létrehozhatja saját .env fájljait. Ezért fontos, hogy ezt a példafájlt is módosítani kell, amikor új konfigurációs értékeket adunk hozzá az .env fájlhoz. Az .env fájl értékei a konfigurációs könyvtár különböző fájljaiból származnak.

Nézzük meg a config mappán belül a database.php-t.



Az env helper metódus ennek a kulcsnak az értékét adja vissza az .env fájlunkból. A második argumentum értéke alapértelmezett értékként szolgál arra az esetre, ha a kulcs nem található. Ea konfigurációs értékeket érhetjük el az alkalmazásunkban.

Nézzük például az app.php-t.

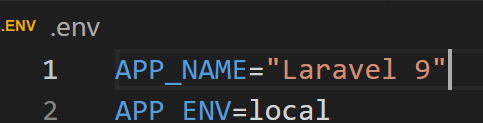


Tegyük fel, hogy szükségünk van az alkalmazás nevére. Keressük meg a Welcome.blade.php oldalunkat, és itt tegyük fel, hogy ennek az alkalmazásnak a címét – a title-t - szeretnénk lecserélni az alkalmazásunk nevére, amely itt található az .env fájlban az app.php fájlon keresztül az env helper segítségével.

Ezt itt úgy kapjuk meg, hogy a config helper módszert használjuk, mert most ez a konfigurációsértékünk. És itt a pont szintaxist használhatjuk: (kb 7-ik sorban: )

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

az app.name-ből az app a konfigurációs fájl neve, a name pedig ez az érték itt. Nézzük a böngészőt, frissítsünk rá, és látjuk a Laravel felirat ott marad.

Változtassuk meg Laravel 9-re, az .env –fájban, de figyelj arra, hogy ezt itt már dupla idézőjelbe kell tenni, mert van benne szóköz!

Frissíts a böngészőben a kezdő oldalra, és láthatod Laravel 9-re változott.

# Adatbázis kapcsolása

Az .env fájlban láthatjuk a kapcsolati adatokat is: láthatod, hogy az adatbázis név is hozzá lett adva, készen áll a kapcsolódásra, ahogyan az a homestead.yaml fájlban kell megadni, ha virtual boxot használsz homestead+vagrant+putty kombóval, de természetesen át írhatod.

Többféle adatbázis kezelőt is használhatunk, pl tableplus, vagy adminer.

Töltsük le az adminer.org oldaláról a legelsőt, ez egy php fájl lesz.

https://www.adminer.org/en/

érdemes a legelsőt letölteni a public mappába kell bemásolni a futáshoz

Az alábbi beállítások az virtual bokszra vonatkoznak, Laragonban nincs más dolgod, mint bemásolni a laragon mappájába és felvenni magad mint új felhasználó. Az oldal neve/adminer oldalon fogod elérni az adatbázist.

Virtual boksz + vagrant +homestead +putty esetére:

DB\_CONNECTION=mysql

DB\_HOST=192.168.56.56 //csak a virtual boksz használtánál vagranttal és homesteaddel

DB\_PORT=3306

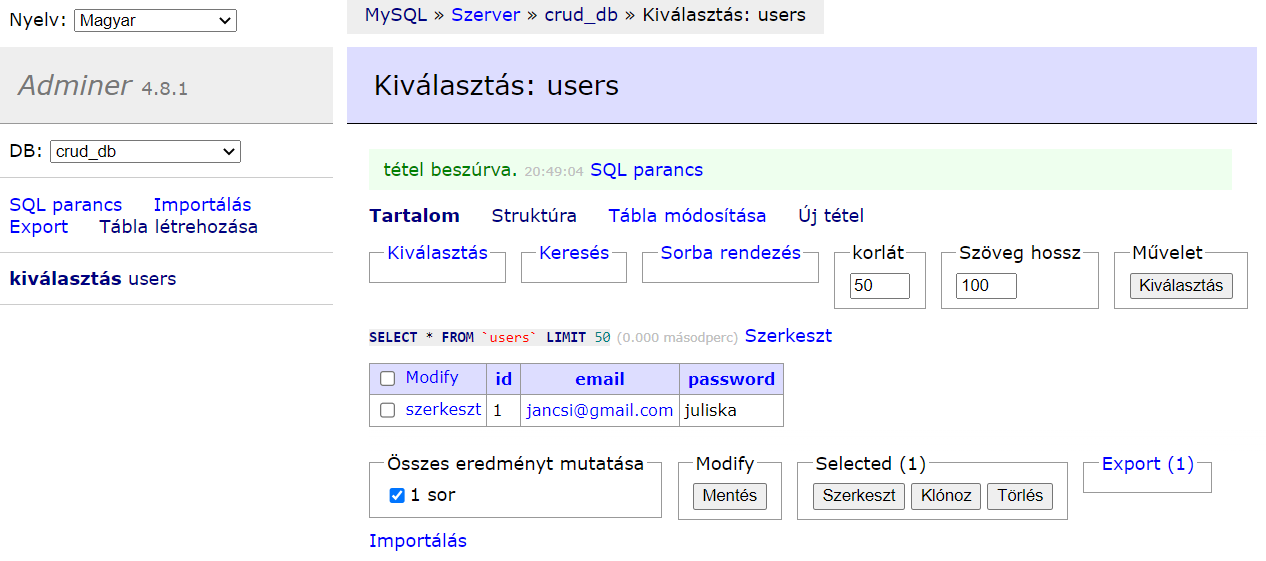
DB\_DATABASE=foglalas\_db

DB\_USERNAME=homestead //homestead-el

DB\_PASSWORD=secret //homestead-el

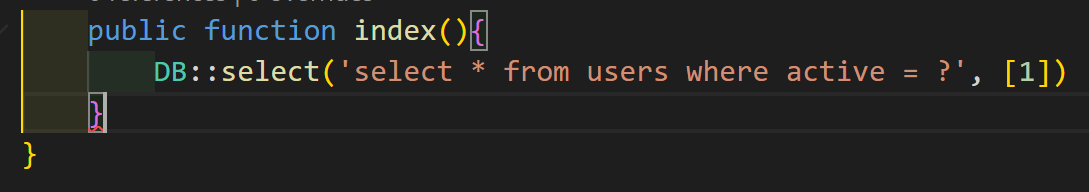
ha nem adunk meg adatbázis nevet, mindent betölt: http://crud.test/adminer

Hozzunk létre benne egy egyszerű adatbázis táblát, id, email, jelszó kombinációval, szúrjunk is be egy sort.



Kérdezzük le az adatbázis adatokat. Nyissuk meg a WelcomeController.php-t. (A Laravel goto view extension segíthet megtalálni, ha beírod a keresőjébe)

A felhasználók lekéréséhez az adatbázisból három különböző módszert használhatunk a Laravelben: **1.) nyers SQL lekérdezések használatával – raw SQL queris**



a függvénybe beírjuk DB::és elkezdjük írni, hogy select a laravel snippet segít, hogy mit illeszthetünk be, persze így még hibát fog jelezni, adjunk hozzá egy újabb extensiont a VSC-hez: PHP Namespace Resolver. Kattintsunk a DB-re jobb egérgomb és import class. Automatikusan hozzá fogja adni a kb 6-ik sorhoz: use Illuminate\Support\Facades\DB;

diedump : dd() -> hogy lássuk mi az eredmény ez rajzolja ki a tömböt, a laravel készítette, a procedurális php-ben csináltuk, hogy dump-al pl var\_dump-al kiírattuk a változót, és ha el akarjuk akasztani ott a kódot akkor a die-al lehet, és a laravelben erre alkották meg a dd-t

javítsuk át így a WelcomeControllerben:

public function index(){

$users = DB::select('select \* from users');

dd($users);

}

web.php-ben a routing:

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

Virtual boxban: ha most próbáljuk meg a crud.test-ben megnyitni még hibát ad, mert a .env fájlban be kell állítani a fejlesztői környezethez a megfelelő adatokat:

DB\_CONNECTION=mysql

DB\_HOST=localhost

DB\_PORT=22

DB\_DATABASE=crud\_db

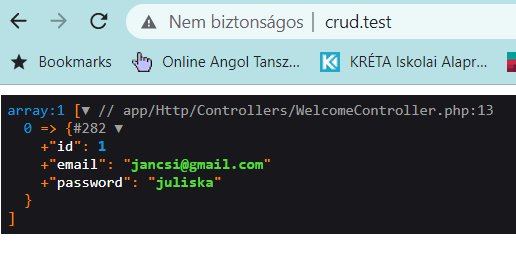
DB\_USERNAME=homestead

DB\_PASSWORD=secret

másold át az exmaple-be is.

Laragonnal jó lesz azonnal:

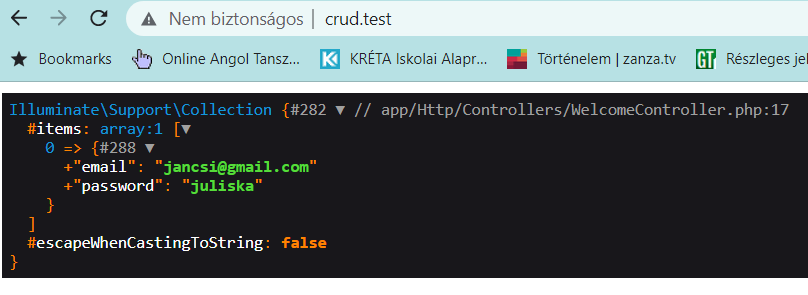
A böngészőben megjelenik a lekérdezés:



**2.) a lekérdezéskészítő használata és - query builder (kommentezd ki az előző megoldást)**

$users = DB::table('users')->select(['email', 'password'])->whereNotNull('email')->orderBy('email')->get();

dd($users);



az eredmény jobb, egy egész gyűjtemény, nemcsak egy tömb

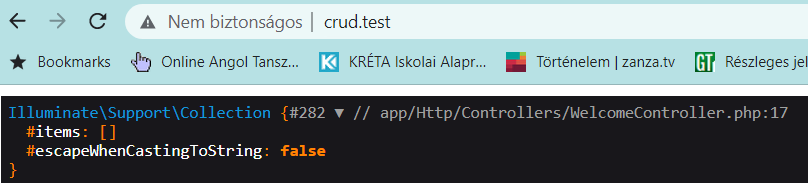
3**.) Az Eloquent ORM** idétlenül lefordítva ékesszóló ORM ☺.

# Migrációk létrehozása és futtatása

Létrehoztunk egy felhasználói táblát egy grafikus felhasználói felület segítségével, de csapatmunkában és az alkalmazás éles környezetben történő üzembe helyezésekor nem a legjobb módszer egy ilyen tábla manuális létrehozása. A Laravel migrációi segítenek a táblaszerkezetek létrehozásában és módosításában, és úgy működnek, mint egy verzióvezérlő. Most töröljük ezt a felhasználói táblát, és hozzuk létre újra áttelepítéssel. (tábla->eldob)

Puttyon írjuk be, laragonnál a terminálon írjuk be:

php artisan make:migration create\_users\_table



A WelcomeController miatt keresi az e-mailt ezért míg ellenőrizzük, javítsd erre vissza:

$users = DB::select('select \* from users');

dd($users);

Vagy előbb egészítsd ki a database migration mappában létre jött fájlt 2 sorral, a users táblában:

public function up()

{

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

$table->id();

$table->string('name');

$table->string('email');

$table->timestamps();

});

}

Amikor egy migrációs tábla létre jön, és migráltuk, akkor fontos, hogy utána soha ne nyúljunk bele, és ne javítsunk át benne semmit. Ha bármit változtatni szeretnénk azt új táblában kell megtennünk.

Minden migrációs táblában két függvény készül egy up és egy down, az up létrehoz, a down töröl.

Két sorral egészítsük ki a name és email-el. majd kiadhatjuk a

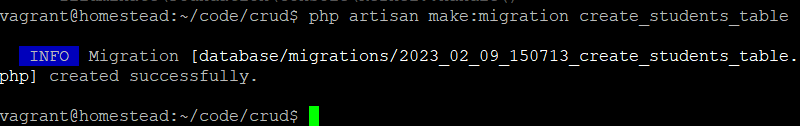
php artisan migrate

Létrehozza a táblát a kért oszlopokkal.

A Laravel Schema osztály biztosítja a táblák kezelését. Új adatbázistábla létrehozásához a következő Schema::create módszert kell használni. A create metódusnak átadott első argumentum a tábla neve, a második pedig az, Closure amely egy objektumot kap, Blueprint, amely felhasználható az új tábla meghatározásához. <https://laravel.com/docs/4.2/schema> tulajdonképpen a tábla szerkezetét mutatja meg nekünk. Két oszlopot láthatunk benne alapértelmezetten, amelyeket a laravel felajánl, az id-t és az időbélyegzőt. Az id az primary key, amit autoincrement beállítású. A timestamp metódus két oszlopot hoz létre, a létrehozás ideje és a módosítás ideje. A kettő közé mi is létre hozhatunk metódusokat, mint a sztring metódus name és email.

Új táblát úgy tudunk létrehozni, ha a puttyon kiadjuk a :

php artisan make:migration create\_students\_table parancsot, és ki is írja nekünk milyen fájlt generált.

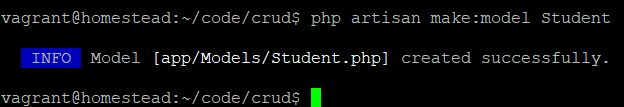
 a migration mappában létre is jött.

Két metódus jön létre a fájlban az up metódus táblákat vagy oszlopokat hoz létre. A down metódus megszünteti azokat. majd: php artisan migrate látható, hogy a migráció fut. Ha ráfrissítünk a böngészően a <http://crud.test/adminer.php> oldalra láthatóvá válik, hogy megjelent az új students táblánk, amelyben létrejöttek az általunk megadott oszlopok.

# Laravel Eloquent – ékesszóló :D laravel

ez a technika megkönnyíti az adatbázissal való munkát. A modellek objektumok, amik leírják az adatbázisod sémáját. Egy objektum egy rekordnak felel meg. Minden tábla társítva van az eloquent modellhez. Segítségével kinyerhetünk, beszúrhatunk, módosíthatunk és törölhetünk adatokat az adatbázisunkba.

Hozzunk létre egy új modellt: php artisan make:model Student

 az **app models** mappákon belül létre jött a **students.php** fájl amiben létre hozta a program a Student classt. Ez az amire szükségünk van, hogy műveleteket végezzünk az adatainkkal.

Szúrjunk be az students táblánkba legalább két sort, két tanulót. Nyissuk meg az **app http controllers** mappákon belül a **WelcomeController.php** fájlunkat. Egészítsük ki az index metódusunkat:

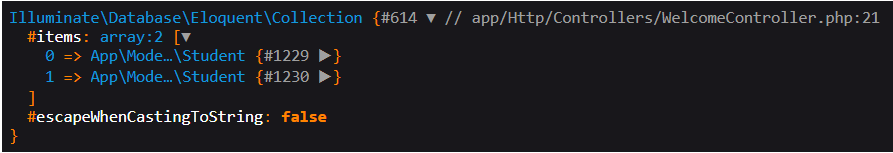
$students = Student::all();

dd($students);

ahhoz, hogy ne adjon hibát a kód elején meg kell adnunk:

use App\Models\Student;

ha most megnézzük a kezdő oldalt, akkor két tanulót láthatunk a tömbben:



Kommentezzük ki a dd()-t és egészítsük ki a fájlt, egy foreach-el iteráljunk végig a tömb elemein:

foreach($students as $student){

echo $student->name. "<br>";

}

Kommentezd ki a foreach-et.

A $students = Student::all(); a Student tábla összes elemét lekérdezi. De így is lekérdezhetjük:

$students = Student::select(['name', 'email'])->whereNotNull('email')->orderBy('name')->get();

dd($students);

Ugyanúgy megkapjuk mindkét elemet a tömb elemeként. Kommentezzük ki.

Próbáljuk ki az **Eloquent**-et. Hozzunk létre ugyanitt először egy objektumot.

$student = new Student();

$student->name = "New";

$student->email = "email@gmail.com";

$student->save();

Azzal hajtjuk végre, hogy újra betöltjük a crud.test oldalt. A böngészőben ugyan nem fogunk változást látni, csak az üres oldalt, de ha megnézzük az adatbázis táblánkat, akkor látni fogjuk, hogy egy új tanuló jelent meg benne, sőt ha többször betöltjük a crud.test oldalt minden alkalommal újra és újra be fogja szúrni nekünk az új tanulót az adatbázisunkba. Láthatjuk, hogy egy új sort szúrtunk be a Laravel's Eloquent ORM segítségével. Most már rendelkezünk a Lateral alapvető ismereteivel, hogy elkezdhessünk egy egyszerű projekten dolgozni.