

Webes ismeretek - Laravel

Támba Bence

tamba.bence@gmail.com

Tartalom

[Haszonos linkek 3](#_Toc55737449)

[Kezdeti lépések 4](#_Toc55737450)

[Fejlesztő környezet telepítése (windows) 4](#_Toc55737451)

[Hasznos parancsok 4](#_Toc55737452)

[Mi micsoda, mi miért van és mit hol találunk? 5](#_Toc55737453)

[Routing 5](#_Toc55737454)

[Controllers 5](#_Toc55737455)

[Views 5](#_Toc55737456)

[Models 5](#_Toc55737457)

# Haszonos linkek

Laravel dokumentáció: <https://laravel.com/docs/7.x/installation>

Xampp: <https://www.apachefriends.org/hu/index.html>

Composer: <https://getcomposer.org/>

VSCode: <https://code.visualstudio.com/>

Bootstrap: <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

**Package gyűjtemény a Github-on**

**Probléma esetén legjobb barátaink a Stackowerflow és a Google**

# Kezdeti lépések

## Fejlesztő környezet telepítése (windows)

1. Xampp telepítése
   1. Amikor azt kérdezi, hogy hozzáadjuk-e a php—t a környezeti változókhoz, arra nyomjunk igent. Ha ezt elmúlasztottuk, akkor itt kapunk megoldást a problémára: <https://dinocajic.medium.com/add-xampp-php-to-environment-variables-in-windows-10-af20a765b0ce>
   2. Telepítés után az apache és mysql elindítását követően a böngészőből elérhető a phpmyadmin (localhost/phpmyadmin)
2. Composer telepítése
3. VSCode telepítése
4. Laravel projekt létrehozása
   1. Terminal (Git bash, powershell, cmd, stb.) megnyitása
   2. A C:\xampp\htdocs mappába elnavigálás, majd a következő parancs beírása
   3. composer create-project --prefer-dist laravel/laravel:^7.0 **<megnevezes>**
   4. Még mindig terminal: belelépük a projekt mappájába és kiadjuk a következő parancsot
   5. php artisan serve
5. A terminal által kiírt linkre kattintva megérkezik a Laravel kezdőlapja, kezdődhet a fejlesztés!

## Fejlesztő környezet telepítése (MAC)

<https://www.youtube.com/watch?v=CbDTwXnAPSk&ab_channel=StackDevelopers>

**Laravel webfejlesztés során a terminal a barátunk, rengeteget könnyít az életünkön**

## Hasznos parancsok

* composer install
* php artisan serve
* php artisan make:controller
* php artisan make:model
* php artisan make:migration
* php artisan migrate
* php artisan migrate:fresh

# Mi micsoda, mi miért van és mit hol találunk?

## Routing

A különböző oldalak és requestek url címe, ahol az alkalmazás várja a klienstől a kapcsolódást

**Helye: routes\web.php**

## Controllers

Az controllerek az üzleti logikát tartalmazzák, ők kérdezik le a modellektől az adatokak, amiket később az egyes nézeteknek (view) visszaadnak, hogy megjelenjenek a felhasználóknál, illetve a felhasználói interakciók itt kerülnek kezelésre, a formok itt kerülnek feldolgozásra, stb**. A controllerek nem nyúlnak az adatbázisokhoz, ők modelleket hoznak létre.**

**Helye: app\Http\Controllers**

## Views

Ezt látja a felhasználó, itt építjük fel a html webes oldalakat. A view mindig egy controllerből hívódik meg, onnan érkeznek hozzá a feldolgozott adatok. A view-n történő felhasználói interakciót (pl.: form posztolása) mindig egy controllernek küldi, ami feldolgozza a beérkezett adatot és visszairányít egy újabb view-ra. **A view-k nem nyúlnak a modellekhez, ők a controllerekkel kommunikálnak**

**Helye: resources\views**

## Models

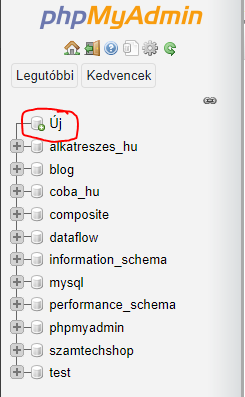
A modellek kommunikálnak az adatbázis különböző tábláival, és leképezik a bennük lévő adatsorokat objektumokká. **A modellek nem hívnak view-kat, ők az adatbázissal dolgoznak együtt.**



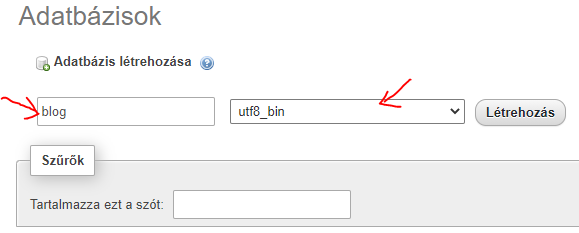
# Első model létrehozása

## Adatbázis létrehozása

1. xampp elindítása, fusson az **apache** és a **mysql** is
2. Böngészőből a **localhost/phpmyadmin** felületen elérjük az adatbázis kezelőfelületét
3. Bal felül hozzunk létre egy **új** adatbázist



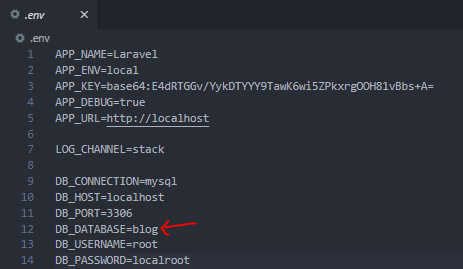
1. Adjunk meg neki egy **nevet**, illetve válasszuk ki a listából az **utf8\_bin** kódolást, és a **létrehozás** gombbal létrejön az új adatbázisunk, egyelőre üresen.



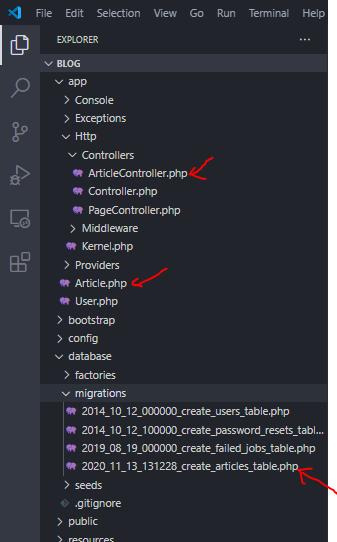
1. Ez után bezárhatjuk a phpmyadmint és a **VSCode**-ban folytatjuk a munkát

## Adatbázis inicializálása

1. A projekt mappájában lévő .**env** file tartalmazza a környezeti változókat, itt definiáljuk az **adatbázis** **kapcsolatot** is



1. Az adatbázis **nevét** mindenképpen szükséges átírni arra, amit az imént megadtunk. Az alapértelmezett felhasználónév **root**, alapértelmezetten a jelszó pedig **üresen** **marad**

Ezzel létrehoztuk, és a programunk számára elérhetővé tettük a fejlesztés során használt adatbázist.

## Model létrehozása

A **Model** a keretrendszer azon eleme, ami az **adatbázissal** kommunikál, és objektumokat generál az egyes rekordokból. Ez teszi, hogy nem kell majd SQL-ben írt sorokkal csúnyítani a kódunkat, hanem elegáns **objektumorientált** megközelítéssel tudjuk használni az adatbázisban lévő adatainkat.

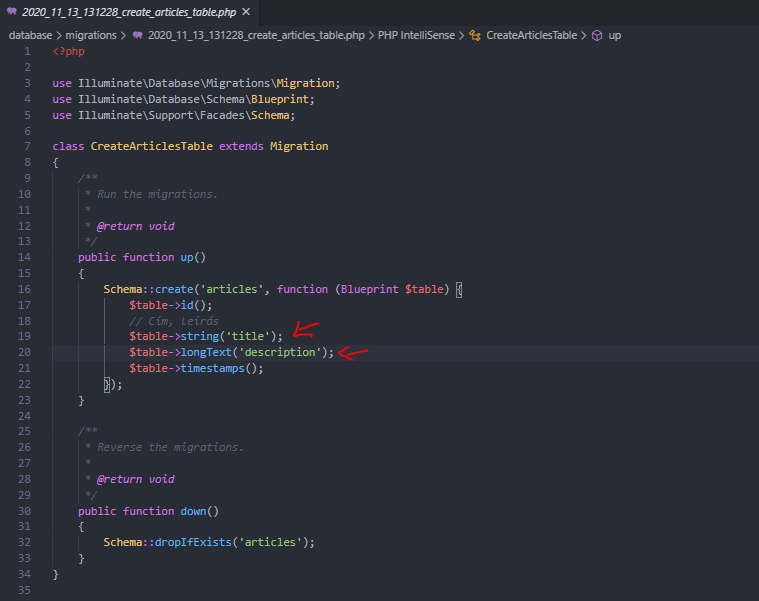
**php artisan make:model Article -mcr**

A parancs alapvetően létrehoz egy **modelt** az app mappában. A -m kapcsoló létrehoz egy **migrációs** filet a database/migrations mappába, a -c kapcsoló létrehoz egy **kontrollert** az app/http/controllers mappába, a -r kapcsoló pedig előre feltölti a **kontrollert** az alap **CRUD** (Create, Read, Update, Delete) funkciók sablonjával.

## Migrációs file elkészítése

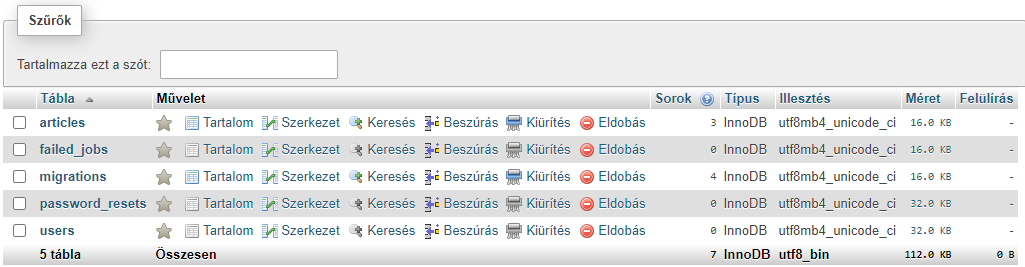
A migrációs file-ok segítségével tudjuk az adatbázisunkat módosítani, táblákat létrehozni, módosítani törölni.

Az újonnan létrehozott migrációs file-unkba hozzuk létre a mezőket. Egy blog cikknek lehet egy címe és lehet egy leírása, ezt egy-egy mezőben fogjuk eltárolni az articles táblánkban.



Ha ezzel megvagyunk, mentés után a terminalban kiadjuk a következő parancsot, mely eredményeként a még nem futtatott migration fájljainkban megfogalmazott változások bekerülnek az adatbázisba.

**php artisan migrate**



# CRUD (bread)

Egy CRUD funkcionalitás kialakítása viszonylag összetett feladat. Ahhoz, hogy bizonyos modelljeink adatait teljeskörűen tudjuk kezelni, minden egyes ehhez tartozó metódust ki kell dolgoznunk, és minden metódusnak minden előfeltételét ki kell elégítenünk.

CRUD: Create, Read, Update, Delete

BREAD: Browse, Read, Edit, Add, Delete

## Előfeltételek

### Routing (web.php)

Ahhoz, hogy a metódusaink működjenek, szükségünk van arra, hogy a program tudja melyik route-on milyen request-eket várjon, és azokat melyik kontroller melyik metódusa fogja majd feldolgozni. A route-ba írt *{id}* egy dinamikus változó, ezt a *web.php* átadja a kontroller hivatkozott metódusának mint attribútum.

Szükségünk van minden egyes metódushoz egy route-ra:

* Listázás – get (index)
* Létrehozás – get (create, itt még csak az űrlapunkat látjuk)
* Megtekintés – get (show, itt a model adatait látjuk, és csak annak az egy modelnek)
* Létrehozáskor eltárolás – post (store, ide megy a create metódus formja, és itt dolgozzuk fel a beleírt adatokat)
* Szerkesztés - get (edit, ez szintén egy űrlap, de már kitöltve az adatokkal)
* Frissítés – put (update, hasonló, mint a store, csak itt meglévő adatbázis rekordot módosítunk, így tovább kell küldeni a route-ban a modelll id-ját is)
* Törlés – delete (destroy)

// Articles

Route::get('/blog', 'ArticleController@index');

Route::get('/blog/create', 'ArticleController@create');

Route::get('/blog/{id}', 'ArticleController@show');

Route::post('/blog/store', 'ArticleController@store')->name('blog.store');

Route::get('/blog/{id}/edit', 'ArticleController@edit');

Route::put('/blog/{id}/update', 'ArticleController@update')->name('blog.update');

Route::delete('blog/{id}/delete', 'ArticleController@destroy')->name('blog.destroy');

## Controller metódusok

### index

A teljes adatbázist lehívjuk az adott modellen keresztül, és továbbadjuk egy view-nak ahol megjelenítjük

public function index()

{

// Lekérjük a Modelt

$articles = Article::all();

return view('articles.index')->with(compact('articles'));

}

### show

Itt lekérjük a route-ban kapott id alapján a szökséges modellünket, és szintén átadjuk egy view-nak ahol megjeleneítjük.

public function show($id)

{

$article = Article::find($id);

return view('articles.show')->with(compact('article'));

}

### create

A create metódus egyszerűen betölt egy oldalt, ahol a létrehozáshoz szükséges űrlap található, ami majd a store metódusnak postol

public function create()

{

$categories = Category::all()->pluck('name', 'id');

return view('articles.create')->with(compact('categories'));

}

Jelen példa már tartalmaz kategóriákat is, amiket átadunk a view-nak későbbi felhasználásra.

### store

A store metódus felel a modell példányosításáért, a form adatainak beletöltéséért és a model mentéséért az adatbázisba. Itt szükséges az adatokat backenden validálni.

Az attribútumként kapott *$request* változónk a küldött form összes adatát tartalmazza

**\*debugging tool: dd($request);**

public function store(Request $request)

{

$article = new Article();

$article->title = $request->title;

$article->description = $request->description;

$article->category\_id = $request->category;

$article->save();

return redirect('/blog')->with('success', 'Sikeres feltöltés');

}

### destroy

A destroy függvény az attribútumként kapott id alapján kikeresi az adott modellt az adatbázisból és kitörli

    public function destroy($id)

    {

        $article = Article::find($id);

        $article->delete();

        return redirect('/blog')->with('success', 'Sikeres törlés');

    }

A destroy metódus delete requestet vár, így a formba ami ide postol bele kell tenni a **@method(’DELETE’)** blade snippetet.

### edit

Az edit az attribútumként kapott id alapján kikeresi a számunkra szükséges modellt és meghívja a formot tartalmazó view-t, aminek átadja a kikeresett modellünket, hogy előre ki tudja tölteni a form egyes mezőit.

public function edit($id)

{

$article = Article::find($id);

$categories = Category::all()->pluck('name', 'id');

return view('articles.edit')->with(compact('article', 'categories'));

}

Az edit form az update metódusnak postol és PUT metódus használ. Ezt a formban jelölni szükséges a submit gomb fölötti **@method(’PUT’)** blade snippet segítségével. Ezen felül a route-hoz hozzá kell tenni a form action részénél a módosítandó model ID-ját is, hogy ezt is át tudja adni az update-nek.

### update

Az update hasonlóan működik, mint a store, viszont nincs medel létrehozás, hanem meglévő modellel dolgozunk. Ezt attribútumként kapja meg a metódus a request-ünkkel együtt, mai pedig a form adatait tartalmazza.

public function update(Request $request, $id)

{

$article = Article::find($id);

$article->title = $request->title;

$article->description = $request->description;

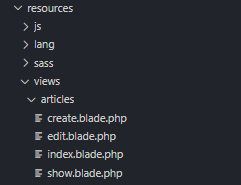
$article->category\_id = $request->category;

$article->save();

return redirect('/blog')->with('success', 'Sikeres módosítás');

}

## Viewk

Alaphangon 4 view-ra lesz szükségünk a *resources/views* alatt. Érdemes csinálni egy mappát, jelen példában ezt *articles*-nek hívjuk.

### Create

Form a szükséges mezőkkel, **@csrf** a *<form …>* nyitótag alá kötelező. A form a **store** route-jára **postolja** az adatokat.

### Edit

Form a szükséges mezőkkel, **@csrf** a *<form …>* nyitótag alá kötelező. A form az **update** route-jára **postolja** az adatokat. Az adatok előre ki vannak töltve a kapott model alapján. Az **update** route **PUT** requestet vár, úgyhogy a **submit** gomb fölé a **@method(’PUT’)** kötelező.

### Index

Tetszőlegesen listázzuk a kapott gyűjteményt (táblázat, kártyák, lista, stb. bootstrap segít ötletelni)

### show

Adott cikk adatainak megjelenítése szintén tetszőleges elrendezésben. Bootstrap itt is segít.